SPATIAL PROVENANCE: A CASE STUDY ON GEOLOGICAL CARBON SEQUESTRATION

BY

CHRISTOPHER P. KOROSE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Geography
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Associate Professor Shaowen Wang

ABSTRACT

The collection, preservation, and accessibility of detailed lineage information derived from the handling of spatial data—spatial provenance—requires effective management of transformational workflows and associated metadata for workflow components. The spatial provenance model presented in this thesis captures important information for understanding spatial data lineage and related data transformation tasks that are chained together in operational workflows. This research has led to the development of an integrated solution for capturing and managing spatial provenance by interfacing with, and building upon, existing geographic information systems (GIS) software. The challenge of organizing, storing, and ultimately accessing spatial provenance information is tackled by exploiting semantic representations and database technologies to enable the comparison of alternative and/or historical iterations of spatial data processing workflows.

Using a case study based on geological carbon sequestration research, this thesis examines source data revisions and updates to a spatial analysis over time. Experiments in tracking the spatial propagation of potential errors and comparing temporal attributes of alternative workflows of processing spatial data verify the effectiveness of the spatial provenance model and its implementation based on semantic representation and query. The case study and experiments serve as a proof-of-concept scenario to demonstrate the benefits of a provenance-enhanced spatial analytical workflow framework. This spatial provenance approach is built upon general components and principles of GIS software that can be scaled to handle massive data sets and support collaborative workflow tasks.

ACKNOWLEDGMENTS

I am most grateful to Dr. Shaowen Wang and Dr. Anand Padmanabhan for their direction in this research, and for their dedication and guidance during the preparation of this thesis. I would like to thank the following people who also provided thoughtful review and helpful comments on this thesis: Dr. Sara McLafferty, Dr. Hannes Leetaru, Dr. Robert Finley, Sallie Greenberg, Dr. Scott Frailey, and members of the CyberInfrastructure and Geospatial Information Laboratory—to whom special thanks are offered for welcoming me into their positive, and motivating, group.

In addition, I would also like to thank the Illinois State Geological Survey, the Midwest Geological Sequestration Consortium, and the U.S. Department of Energy – National Energy Technology Laboratory for the spatial datasets and calculation methodology as used in the geological carbon sequestration case study.

Finally, I owe the completion of this work to the love and continued support I have received from Melony Barrett, family, and friends from work, school, and elsewhere—every bit of encouragement, no matter how large or small, can, does, and did make a difference. Thank you all, so much.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The quantity and quality of spatial (geographically-referenced) data continue to increase, driven by rapid technological advances and massive needs in numerous application domains. As the field of geographic information science (GIScience) has entered into a data-intensive era (Wang 2010), efficient handling and intelligent analysis of spatial data have become increasingly important. Data handling and analysis, in turn, generate more data—i.e. 'data about data'—also known in the literature as metadata. Inasmuch as the management and documentation of data are important, so is data provenance—i.e. tracking the origins of data sets, and processes by which they are transformed to their current form (Buneman et al. 2001, Wang et al. 2008). In real-world data processing scenarios, results are often reassessed and updated. The valuable information outlining the lineage of existing results can be (and is often) easily lost unless a systematic effort is undertaken to record, update, and share provenance information. The issue of data provenance is relevant to numerous scientific fields of study, and this research focuses on data provenance related to geographic information. The purpose of this research is to advance knowledge of provenance for spatial data sets by demonstrating the benefits of adding provenance capabilities to spatial analysis and associated data transformations based on Geographic Information Systems (GIS).

Spatial data in the context of GIS and associated applications, derived from one or more data sources, are often transformed into information and knowledge. Although rudimentary mechanisms exist within current desktop GIS software for recording historical elements of a dataset's creation, the collection and preservation of detailed lineage information for spatial data—spatial provenance (Wang et al. 2008)—is often lacking in informative detail about transformational workflows, i.e. how spatial data are

transformed. Therefore, it is crucial to develop a spatial data processing framework that will provide convenient mechanisms to record and manipulate detailed and informative spatial provenance.

Spatial provenance involves the capture and organization of data processing parameters, transformational workflow sequences, and information and assumptions related to spatial data (raw or intermediate)—as well as changes in any of these—and is useful to illuminate spatial transformation processes and intermediate data that can be made available to end users for discovery of new geospatial information and knowledge. Making spatial provenance information available to data producers and consumers also serves as an aid to 'transparency' in GIS data by fostering a better understanding of derived data and the transformational processes used to obtain spatial analysis results, thereby helping minimize the possibility of errors being introduced during data transformations. Furthermore, provenance management capabilities in GIS aid in the comparison of workflow iterations based on differences in data inputs, scopes of study, spatial relationships, and data-transformation procedures, and are likely to contribute to the development of novel geospatial knowledge.

In this thesis, a generic framework is established for collecting and managing spatial provenance based on desktop GIS software. An integrated strategy was developed for managing provenance information by interfacing with, and building upon, existing GIS information management components. The case study and experiments focus, in particular, on overcoming the limitations of desktop GIS software, namely, Esri's ArcGIS (Esri 1999). The challenge of organizing, storing, and ultimately querying provenance information is tackled by exploiting open-source database technologies found outside the GIS software itself, which allows for flexible storage options and powerful queries of linked information. The effectiveness of this implementation to provide valuable insights to researchers is assessed in a case study by employing the spatial provenance framework

during the spatial data operations pertinent to carbon sequestration research in the geological domain (Finley 2005, U.S. Department of Energy [DOE] 2007).

In the case study, the volumetric estimate of carbon dioxide (CO_2) sequestration potential (or CO_2 storage resource) for a subsurface geologic reservoir is calculated. This estimate is periodically revised—typically due to new spatial data that have been obtained, consideration of different calculation methods or scales of assessment, refinement of parameter values within the calculations, or any combination of these. The case study in this thesis examines data revisions and updates to a spatial analysis over time, and the experiments show how incorporation and management of spatial provenance help users to track and evaluate the effects of these updates. The case study and examples serve as a proof-of-concept scenario that is used to assess the entire provenance-enhanced spatial analytical workflow framework.

In the chapter that follows, work related to provenance management within GIScience is highlighted, after which the context of a real-world case for managing and using provenance is presented. Next, a model based on the Open Provenance Model (Moreau et al. 2007) is described, to facilitate the collection and management of necessary spatial provenance. The model presentation is followed by a description of a practical strategy to capture, store, and query spatial provenance using GIS and database technologies. An implementation of the strategy is presented, using experimental examples to assess the impact of the automatic collection of a rich set of spatial provenance as used in geological carbon sequestration research—which support the argument that the collection and query of provenance information is beneficial to the advancement of spatial analytical capabilities in this and other fields of study.

CHAPTER 2

RELATED WORK

In this chapter, discussion is focused on provenance-related research in GIScience after a brief overview of the subject of data provenance.

2.1 CONCEPT OF PROVENANCE

The concept of provenance can be traced to research on data lineage that refers to the tracking of data processing (Clarke and Clark 1995, *in* Bose and Frew 2005). Both forward and backward data lineage has been addressed, either as 'child' (descendant) or 'parent' (ancestor) links between data and transformational processes (Lanter 1991, 1993, in Lanter 1994). The notions of prospective and retrospective provenance expand on these ideas, and respectively identify procedure-based workflow information as compared to parameters or settings related to procedures' runtime implementation (Bose and Frew 2005, Zhao et al. 2006, Clifford et al. 2008). In this light, then, one can view the idea of provenance collection as accounting for the information to allow assessment of workflow paths, data transformation, and data quality, namely: data sources, input parameters, transformations applied, intermediate data, analyses, and decisions leading to final data processing results.

Provenance is often not well captured within conventional GIS environments, i.e., is lost, or not presented to end users. However, knowledge of data provenance is important in assessing: quality of information, methods to verify data or correct errors, and ultimately, the "fitness for use" of the derived data for a particular purpose (Veregin and Lanter 1995). In addition, a robust assessment of intermediate data-handling steps may present data producers and consumers alike with analysis options that may lead to results which could be obtained with a higher degree of confidence.

Data provenance has broad applicability to numerous scientific fields of study, and has been researched extensively with respect to shared computation and analysis (e-Science) in the fields of molecular biology and astronomy (Simmhan et al. 2005, Bose and Frew 2005, Renaud 2008, eBank UK 2008). In particular, the processing of spatial data poses interesting challenges related to the collection and effective use of provenance information, and is a growing area of research interest (Bose and Frew 2004, Yue et al. 2009, Wang et al. 2008).

2.2 DATA PROVENANCE WITHIN GISCIENCE

Data provenance research within the field of GIScience is concerned with spatial data lineage and recording spatial transformations applied to data (Lanter 1991). In this section, several research themes are surveyed.

2.2.1 Lineage tracking

Although systematic studies of provenance in GIScience are still in their infancy (Wang et al. 2008, Yue et al. 2009), a formalized geospatial metadata structure that incorporates data lineage has been instituted as specified by the Federal Geographic Data Committee (FGDC)'s Content Standard for Digital Geospatial Metadata (1998), and later, by international standard ISO 19115 (International Organization for Standardization 2003). In addition to documenting elements germane to spatial data such as projection and datum information, horizontal coordinate units, and geographic extent, geospatial metadata (hereafter stated simply as 'metadata') specifies lineage elements for data source(s) and data processing steps as simple lists. However, practical limitations to metadata elements are that they do not provide for "the means by which this information is organized in a computer system or in a data transfer, nor the means by which this information is transmitted" (FGDC 1998, *in* Bose and Frew 2005)—and the degree of

informational detail to record, if any, as well as the timeliness of updates are decisions ultimately left to data producers.

Early research represented spatial data lineage as linked elements and processes using directed acyclic graphs (DAG, Bachman 1969)—a network of nodes and links between these nodes that illustrates directional order (Figure 1). Tomlin and Berry (1979) used the DAG as a basis for their graphical cartographic model—a diagramming convention for the knowledge representation of map elements which gave semantic representation to these nodes and links (*in* Lanter 1991). This not only proved to be an efficient and scalable way to represent 'parent' and 'child' links between mapping

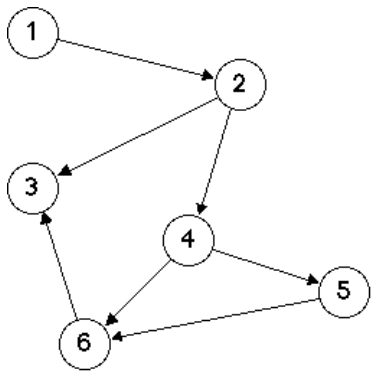

Figure 1 A simple directed acyclic graph (from Keller 1998).

elements and transformational processes (Tomlin and Berry 1979, Tobler 1979), but would serve as a graphical base for the Entity-Relationship (E-R) Model (Chen 1976, 1977)—which extended the DAG to further define properties of specific 'entities' and relationships between these entities. Nyerges applied the E-R Model to the description of spatial properties of cartographic data (1989, *in* Lanter 1991)—an approach suitable to be expanded to include spatial data elements and transformational operations for the automation of lineage data generation, storage, and query (Lanter 1989, 1991, 1993).

In Lanter and Veregin's Lineage Information Program (1990) and later GeoLineus (Lanter 1992, Geographic Designs 1993), based on Arc/Info GIS software (Esri 1982), a program user was prompted to type in information for source data and final data processing products, which was then stored in a 'metadatabase' outside of the GIS software. Arc/Info's *log file* natively captured a historical listing of Arc/Info commands that were run, with various system- or software-generated status messages, etc., but the lineage programs extended the tracking of transformations applied to data by storing explicit linkage between parent and child spatial data files (often referred to as GIS, or map, 'layers'). Thus, the software metadata (e.g. commands, messages, parent and child relationships between GIS data 'layers') differentiated between source, intermediate, and final data, and allowed for searches of the resultant chain of lineage information. In addition, the user could parse Arc/Info command information from the lineage chain into a text file, and then re-run the commands for the purpose of updating data products (Lanter 1991).

Similarly, Spery et al. (1999, 2001) integrated DAG with their spatial data to represent and track geographic changes made to French cadastral information (e.g. subdivision of lots)—boundary adjustment descriptions which were stored in a relational database. The database allowed for historical or targeted queries of the lineage information, but unlike Lanter and Veregin's work, only the current spatial data layer was maintained and linked to the descriptions of geographic changes.

Recent works have focused on automatically collecting and organizing process-level (i.e. procedural) information from linked GIS data transformations. Bose and Frew (2004) used such data transformation information to generate process-level 'lineage metadata' Extensible Markup Language (XML) documents. These were stored and cross-linked to documentation for parent and child data files, and used to search for which particular components were involved in specific invocations of a data handling

workflow. Yue et al. (2009) expanded these ideas to collecting and querying metadata for chained geospatial data web services. Wang et al. (2008) developed a 'provenance-aware' GIS using service-oriented architecture, decoupled spatial and aspatial data stores, Resource Description Framework (RDF, Klyne et al. 2004)-formatted semantic linking of process steps and data, and provenance information query.

2.2.2 Assessment of data quality and error propagation

Ancestry lineage relationships have been used in assessing the propagation of error in spatial data processing through a chain of GIS data transformation steps (Lanter and Veregin 1990, 1992, Veregin and Lanter 1995). Algebraic error-propagation formulas assigned to basic GIS transformational operations were coupled with lineage information, and used to determine data source qualities (or measures of error) needed to reach a certain statistical confidence level in derived output data. Measures of 'proportion correctly classified' (Lanter and Veregin 1992) were assigned to source data and were operated on by the algebraic expressions for specific GIS transformations (e.g., *overlay*, *select data from*, *buffering*, etc.)—a numerical combination of which resulted in the total measure of error for the output spatial data layer. Through tracking the "effects of changes in the accuracy of each source layer" (Veregin and Lanter 1995) one was able to identify which source data, if any, had the most impact on the output. Using the error propagation formulas, target quality enhancements could be identified for influential layers, helping to bring final data products to a desired level of quality (Veregin and Lanter 1995).

2.2.3 Workflow

The automatic updating of spatial data processing results, or layers affected by changes made to source or intermediate data, was a driving factor for early GIS data

lineage research—as derived results are, in effect, obsolete after updates have been made to their data sources. Lanter's programs for Arc/Info automatically performed equivalence tests on stored lineage information to distinguish original data sources from derivations, and provided a means to track and query updates to the entire data lifecycle—e.g. when a data element was last-modified, or if certain temporal attributes were out-of-date with respect to a reference time, etc. Stored software commands and parameters could then be used to automatically traverse the linked workflow processes and re-generate the derived data (Lanter 1994).

A DAG is used in Esri's ModelBuilder (first introduced in ArcView 3.2, Esri 2000a)—a software tool with a graphical user interface for creating and running spatial analytical workflows, i.e. linked data and data-transformation steps, within ArcGIS (Figure 2). ModelBuilder manages data (e.g. input/output paths, allows a user to save or delete intermediate data) and transformations, and can be a useful aid in visually communicating GIS data-handling workflow steps. To some degree, ModelBuilder takes advantage of lineage information in its implementation of 'smart' data updates in that existing intermediate data are not re-created during the execution of any workflow (i.e. 'geoprocessing model') if they are not affected by updates to specific source data elements; however, it is assumed that the intermediate data have been saved, and have not been edited or updated since the geoprocessing model's last execution. An XML-formatted *model report* generated from the ModelBuilder interface lists the names of input and output data and parameters for 'geoprocessing tools' (e.g. ArcGIS commands, or data-transformation tasks, such as *buffer*, *intersect*, *clip*) executed within a ModelBuilder workflow, but the report presents all of the information at once, and does not organize or link the tools into chained data processing steps—thus, there is no

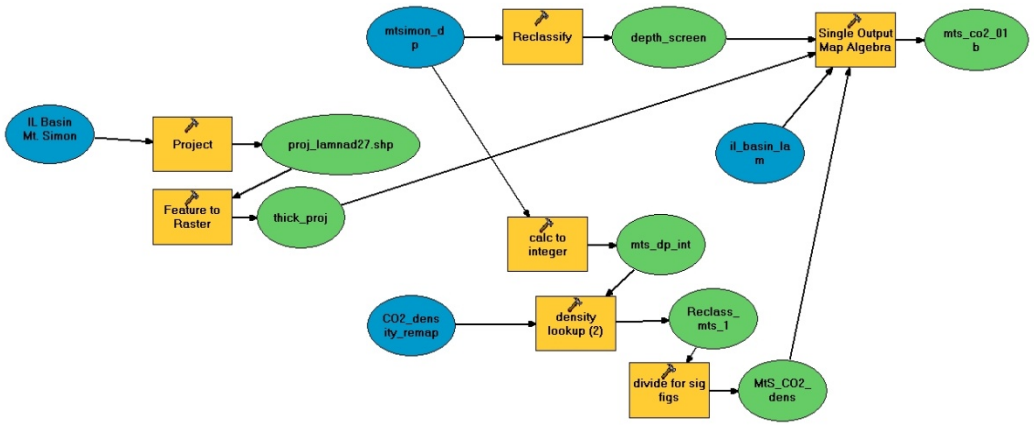

Figure 2 Directed graph representing the spatial analytical **workflow** used for the case study and experiments presented in this thesis. Blue ovals represent source **data** inputs, which are linked to transformational processes, or GIS **tasks** (orange rectangles). Data, or results, subsequently derived from the input data sources and tasks are represented by green ovals; the last green oval in the graph (in upper-right) represents the workflow's final output result. *Image created using Esri ModelBuilder (Esri 2000a).*

systematic way to automatically combine, compare, or query data derivation history from different executions of a geoprocessing workflow.

ArcGIS' *results tab* and *history log files* capture and present to users a record of program actions taken, including but not limited to data-transformation and data-handling tools, parameters, and status information (such as successful completion, execution time, failure to complete, etc.). These reporting mechanisms are similar in function to *log files* in Arc/Info. The information contained in the *results tab* is only stored internally within the software applications, and only a portion of the historical geoprocessing information (such as the most recent tool/command executed, and processing date/time) is automatically posted to the resultant data layer's metadata XML file. Although the information contained within ArcGIS' *results tab* and *history log file* is automatically generated by the software for singular tool executions, each workflow run in the ModelBuilder environment is treated as a single tool, or data transformation process—and detailed and valuable information about the intermediate, combined operations or tasks comprising the entire workflow is not presented in these reports.

ARIS has developed an interesting DAG-based dataflow management program, ArisFlow (2005), designed for managing automated computing processes—which the company claims is useful for, but not limited to, GIS analytical processes or modeling programs. Similar to Esri's ModelBuilder, ArisFlow can selectively execute only the parts of any workflow that may need to be reexamined based on programmatic or data-related changes. Also, data lineage is automatically stored within the DAG-based system, and the program documents the executed operations thus allowing users to reproduce their results (ARIS 2005). Other studies of GIS data or data handling models have addressed the management of data versions, although these have dealt less with typical spatial workflow operations and more with the temporal dimension of editorial/transactional recordings of different internal states, or versions, of a data file (Medeiros and Jomier 1993, 1994, Medeiros et al. 1996, Spery et al. 1999, 2001, Esri 2002, 2007, Vert et al. 2002).

Spatial data workflow systems e.g. GeoOpera (Alonso and Hagen 1997) and ESSW (Frew and Bose 2001), cyberinfrastructure-based GIS—such as GISolve, a TeraGrid GIScience Gateway (Wang and Liu 2009), the GEON project (Geosciences Network 2002), and several examples from the National Science Foundation's Cyber-GIS workshop (2010)—represent overlapping areas of active research. In addition, researchers have sought to understand and standardize the semantics and meanings associated with spatial data ontologies to enable interoperable or collaborative, distributed GIS (Fonseca et al. 2002, Luo 2007). These research directions are closely related to, or interlinked with, that of provenance, in that they help define and refine the standardized base for communication and collaboration, and necessitate data sharing, collaborative workflow systems, and distributed and communal knowledge that are collectively enabled by integrated spatial provenance management.

As the previous sections illustrate, there are several themes within spatial

provenance research that are even more relevant today and to future advances of GIScience. A spatial provenance framework is an important need (Wang et al. 2008, Schuurman and Leszczynski 2006, Gahegan and Pike 2006, Grossner et al. 2008) in fostering better-informed data sharing and communication among collaborators, and in allowing researchers to examine their own and others' data in more meaningful ways. In the following chapters, such a framework is described for managing spatial provenance in a modern desktop GIS environment using a case study in the domain of geological carbon sequestration.

CHAPTER 3

CASE STUDY

This chapter describes a real-world application based on a spatial data transformation workflow in the geological carbon sequestration domain, in order to evaluate the benefits of a spatial provenance framework developed in this thesis research.

3.1 DOMAIN DESCRIPTION

The identification of geologic reservoirs suitable for storing carbon dioxide (CO_2) is of increasing importance as subsurface geological carbon sequestration becomes a potential aid in climate-change mitigation and the reduction of greenhouse gasses released to the atmosphere (Finley 2005, U.S. DOE 2007, 2008). As part of an on-going study, geologic reservoirs are being assessed and modeled, and desktop GIS software is used to estimate the regional volumetric CO_2 storage potential of rock formations. For this study, the basic volumetric equation for estimating a reservoir's pore volume is:

```
Reservoir Pore Volume = Area * Thickness * Porosity                (1)
```

The basic case is extended to estimate the potential CO_2 storage resource (mass, in metric tons) in the reservoir, by including the density of CO_2 and assuming a CO_2 storage efficiency[1] (U.S. DOE 2007, 2008). Cell-based raster data are used to represent scalar field variations of reservoir properties, and are spatially incorporated in the calculation to derive a map layer of the estimated CO_2 storage resource for the geologic reservoir. The spatial analysis involves screening to eliminate unfavorable areas where the reservoir depth is less than 800 meters (van der Meer 1992, U.S. DOE 2008). Thus, the volumetric equation for the geologic reservoir is modified to:

[1] The CO_2 storage efficiency factor represents a fraction of the total reservoir's pore volume that is filled by CO_2 (U.S. DOE 2008).

```
CO2 storage resource, mass = Raster cell area * Thickness * Porosity *
                        (CO2 density: look-up table based on depth) *
                            (Depth_screen: omit areas where depth < 800 meters) *
                               (Storage Efficiency Factor, as a percentage) /
                                  (Unit conversion factor)                  (2)
```

However, oftentimes in a real-world scenario, an analysis or workflow is later reassessed and updated—either due to new data that have been obtained, consideration of a different analytical methodology or different scale of assessment, refinement of parameter values, or a combination of these. Any of these is a valid reason for reassessment—yet when previous results are used to check and verify potential updates, it is often difficult to tell from which set of calculations, or from which set of source data, certain results have been derived. These data revisions and updates to a spatial analysis over time provide the basis for implementing and evaluating this study's spatial provenance model. However, a better understanding of the spatial provenance model's architecture may be gained by first examining a conceptual spatial data model for this case study.

3.2 DATA MODEL

To estimate the amount of CO_2 storage resource that could potentially be sequestered in a subsurface geologic reservoir, map algebra functions are used to operate on raster data layers, which are primarily derived from point-source geologic information such as reservoir thickness or subsurface depth (Figure 3a). Thus, in this case study, the domain data model is data format-driven, and relies on the values in each raster layer being the measure for one particular geologic property that is being regionally mapped (e.g. one raster map layer represents reservoir thickness, another represents reservoir depth, etc. as shown in Figure 3b). The inherent properties of the data used in the case study are based on the underlying raster data structure of spatial data layers (Figure 3c),

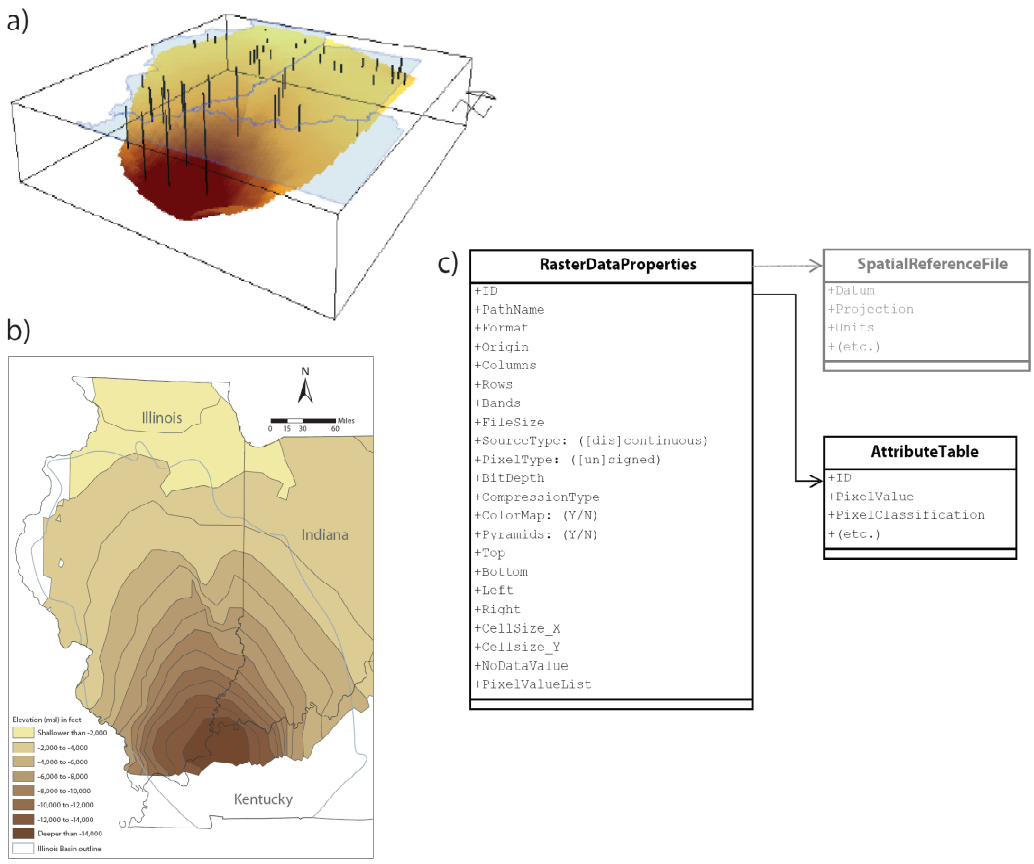

Figure 3 Conceptual data model for regional geologic data to support volumetric assessments of CO_2 storage resource potential, showing: a) 3-D spatial context for reservoir structure and point-source drillhole information, b) representation of regional geologic reservoir as 2-D spatial data (map) layers, and c) internal data structure and properties inherent to spatial data in raster format. *Image sources: a) this thesis, b) Midwest Geological Sequestration Consortium (Finley 2005), and c) compiled and modified from Esri raster dataset documentation (2009).*

and contain attribute information about such elements as: file name and location (*Name*, *PathName*), how the raster pixels are stored (*Format*, *PixelType*, *BitDepth*, *CompressionType*), how the pixels are arranged in map coordinate space (*Origin*, *Columns*, *Rows*), and how these coordinates are registered to real-world locations (*Cellsize* and spatial reference information). Although these data properties are required for digital data creation and interpretation by computer systems and software, it is not necessary to keep track of all of the data structure's internal elements in order to establish an effective provenance management framework. As presented in the following chapter,

this work strikes a balance between important **data**-related information and data-handling **process** information in a model for the management of spatial provenance.

CHAPTER 4

SPATIAL PROVENANCE FRAMEWORK

A spatial provenance framework is built on a conceptual model and general information management architecture, and implemented to integrate components for provenance capture, storage, and query. In this model, provenance information is collected for spatial data and transformational processes (also referred to as tasks) that are chained together and grouped in an operational workflow. The collected information is stored in a database, and can be queried and compared against alternative and/or historical iterations of the workflow. The organization of, and relationships between, relevant task- and data-related informational elements are the foundation for this spatial provenance management model, and are detailed in the following section.

4.1 CONCEPTUAL MODEL

The model for the management of spatial provenance information is based on the Open Provenance Model (Moreau et al. 2007). The Open Provenance Model uses DAG to represent historical records of a workflow's execution, and specifies that the relationships and dependencies between elements of the workflow are precisely defined. In this thesis, lineage and ancillary information are grouped at the functional levels of task, data, and data properties, and are collected from the chained source data, transformational tasks, and derived data. The lineage relationships and associated data- and task-related information are organized and stored in a relational database, and comprise the suite of spatial provenance information. As time progresses and source data and task parameters are updated, the information collected from these processing tasks enriches the spatial provenance associated with the data, exposing the history of changes within the chained tasks—and the GIS processing workflow as a whole—to further study.

This model is designed to: 1) enhance the management and organization of alternative spatial data-handling pathways, 2) support automated organization, derivation, and preservation of detailed lineage information—to prevent these from being lost, 3) allow for analysis and discovery of new knowledge gained from rich provenance information, and 4) enable information dissemination and presentation as coherent, searchable links intended to facilitate the development of new insights and communication in support of collaborative work.

To manage the provenance information, timestamp-based IDs are implemented for every workflow task, thereby ensuring a unique identifier for every instance of a GIS data-handling process that has been run. *SpatialWorkflow* is initiated by a user, and is represented as DAG for chained data-handling tasks (Figure 4). A run-time ID (*Runtime_ID*) is automatically created, but a common name and/or any comments or descriptions pertinent to this particular instance of *SpatialWorkflow* can be input by the user at run-time to facilitate annotation and search. Any *SpatialWorkflow* process is composed of one or more data-handling or data-transformation tasks (*SpatialTask*). Information relevant to the tasks includes, but is not limited to: task common name (*Name)*, the software command along with input parameters issued (*Command*), runtime duration of the data-handling task (*RuntimeDuration*), and numerical counts of data elements viewed as either input to or output from this particular task (*InputCount*, *OutputCount*). Causal relationships, or workflow operations order, such as intermediate or final derived data, can be determined from the designations '*hasInput'* or '*hasOutput'*—which link tasks directly to the data elements. Each task is directly related to its input and output data elements individually, and additional provenance information is captured at the *Data* level.

At the model's *Data* element level, information for the filesystem path/name or

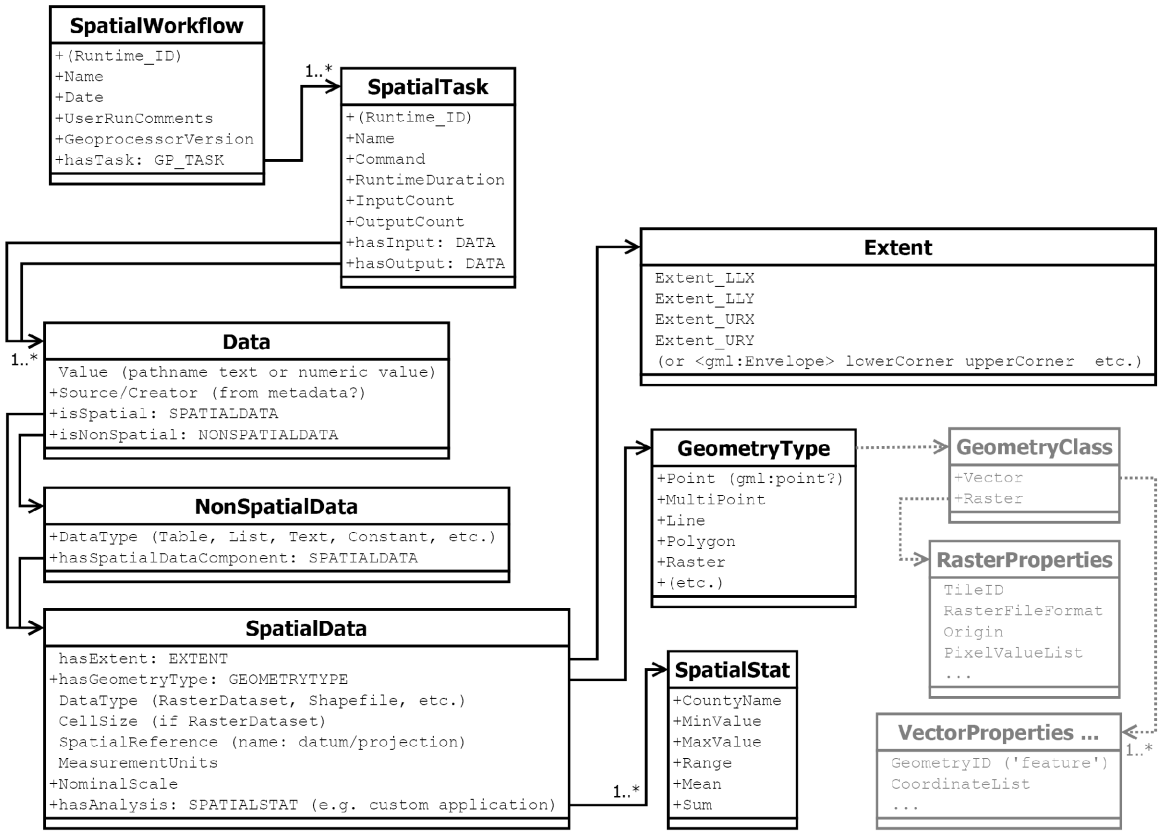

Figure 4 A spatial provenance model. Elements connected by dotted lines (GeometryClass, RasterProperties, VectorProperties) denote transitions from provenance information to internal information managed by GIS software. Element attributes preceded by a plus sign (+) are related to provenance management, and extend beyond the scope of the data model.

other representative value (*Value*) is collected, in addition to other potential attributes, e.g. those related to the data element's *Source* (e.g. agency or owner) or usage context. Data elements are sub-classified as either *SpatialData* or *NonSpatialData* elements. For nonspatial data, an attribute named *DataType* was used as a placeholder for metadata information that needs to be recorded. It may be useful to have a possible mechanism to associate nonspatial data that may be 'joined' to or used in conjunction with spatial data (e.g. via a *hasSpatialDataComponent* attribute for nonspatial data), but the focus of this research is mainly on spatial data.

For spatial data, more complex information is recorded, including spatial data

type (*DataType*), spatial reference (*SpatialReference*), coordinate measurement units (*MeasurementUnits*), and nominal scale of the data layer (*NominalScale*). The *hasExtent* attribute references the minimum and maximum coordinate pairs denoting the spatial bounding area (or 'envelope') of the mapped data (*Extent*). Similarly, via the *hasGeometryType* attribute, the *GeometryType* of spatial data describes the basic spatial organization of the data such as Point, Line, Polygon, etc., and may be based on accepted standards or definitions, e.g. Geography Markup Language (GML, Lake and Cuthbert 2000). *GeometryClass* properties can be further mined to find *CellSize* for data of *Raster* geometry class, or for *Vector* data: the list of geometric coordinate pairs used to delineate the individual geometric shapes (or features) of the spatial data. However, it is important to caution that collecting spatial **provenance** information is of interest in this study, rather than the data internally managed by GIS software.

It is also at the model's *SpatialData* level that customized information about spatial statistics or spatial analysis can be incorporated into the provenance model. In the case study, spatial statistics are performed on the final output dataset, the results of which provide summary information for defined areas or regions of interest. This statistical information is collected and stored in the database along with other provenance information, and can be linked along the workflow lineage chain back to the specific instances of spatial data processing tasks, and all of the associated spatial provenance information.

In the conceptual model presented above, spatial provenance information is represented. When used in a workflow scenario pertaining to the estimation of CO_2 storage potential in geologic reservoirs, this not only helps to keep track of the various parameters that have evolved over the execution history of this data handling workflow, but also enables the assessment of spatial differences in how these alternative calculations affect the volumetric results for finer-grained subsets of a study area.

4.2 ARCHITECTURE AND IMPLEMENTATION

In this section, the architecture of the spatial provenance framework is first described in a general sense, followed by an implementation based on technologies specific to the case study.

4.2.1 Provenance model architecture

The architecture for spatial provenance management, in this thesis, is functionally divided into components for information capture, storage, and query (Figure 5). Conceptually, provenance information is captured within GIS software environments,

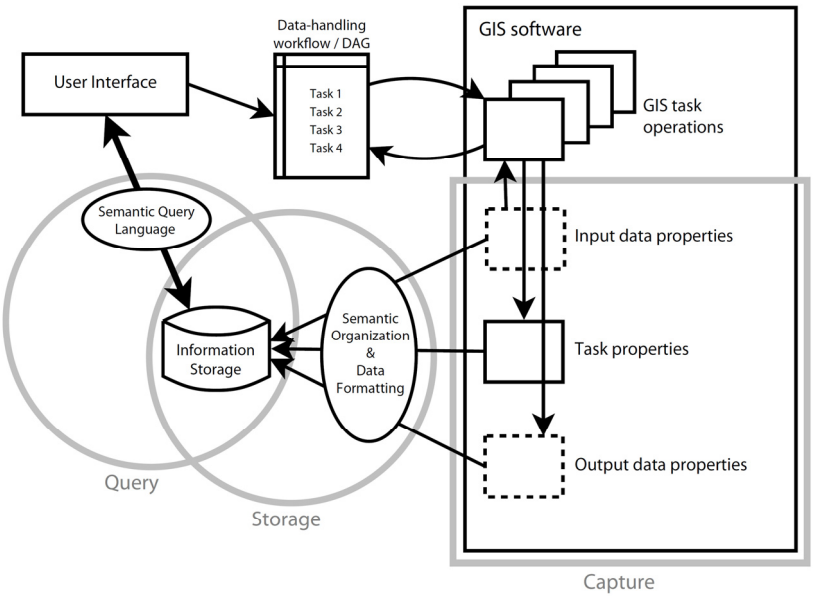

Figure 5 Generalized representation of spatial provenance management architecture.

where programming modules are used at runtime to interact with individual data-handling tasks in the DAG-based workflow. The modules probe the data involved in these tasks for pertinent geospatial information; they also record process information from the task's executed commands and data transformation operations, and record the

status of the data as being either input to, or output from, these specific data-handling operations (refer to Figure 4). The information for chained data-processing tasks is organized into parent-child links, formatted using 'semantic web' technology such as RDF or Web Ontology Language (OWL, Patel-Schneider et al. 2004), and is passed to a database for storage. Users can directly interact with and query the spatial provenance data store using semantic query languages that traverse and interpret the relations between chained data-processing tasks and input/output datasets. This architecture is software-independent, and thus can be implemented to interface with virtually any GIS environment.

4.2.2 Implementation

For the geological carbon sequestration case study, the analytical workflow is performed using ArcGIS—hence, work to implement spatial provenance management was conducted using this software. However, this implementation uses several technologies to effectively capture, store, and query the suite of provenance information (Figure 6). Spatial provenance information is captured using the ArcGIS Geoprocessor. The information for chained data and processing tasks is organized into a DAG using RDF, and stored in a MySQL database (Oracle Corporation 2008). Users can either utilize the user-interface or directly interact with and query the provenance data store using the SPARQL Protocol and RDF Query Language (SPARQL, Prud'hommeaux and Seaborne 2008). The spatial provenance information management framework is implemented in the Python programming language (van Rossum 1995). This method was chosen because of Python's powerful flexibility—it can communicate with all elements of this implementation, being supported natively by the ArcGIS Geoprocessor as well as providing libraries to support user-interface development, RDF data handling

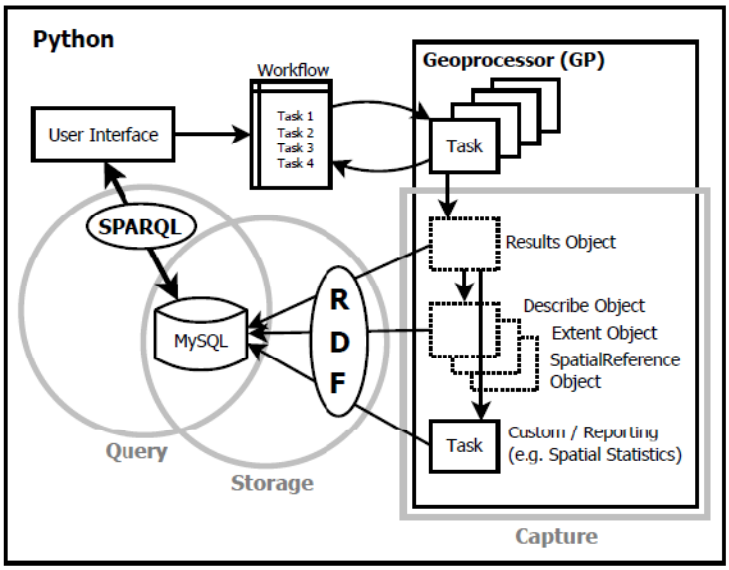

Figure 6 Schematic representation of spatial provenance management implementation.

and SPARQL query (RDFlib), and MySQL database integration.

This implementation exploits the ArcGIS 9.3 Geoprocessor (GP)'s *Result Object* and *Describe Object* (Figure 7), whose associated attributes are only available via the ArcObjects (Esri 2000b) application programming interface (API), and require a customized programming solution for their direct use. The *Result Object* contains properties and methods which are programmatically interacted with, in tandem with other programming objects in the ArcObjects API as well as external to ArcGIS, to a) capture and build upon the GIS software's data-attribute identification and basic history-reporting messages, and b) combine these disparate elements into a meaningful, linked information chain. Thus, the *Result Object* is this implementation's gateway to the collection of spatial provenance information.

The provenance recording component interfaces with the GP *Result Object* during the execution of each GP 'geoprocessing' task, to collect and format provenance information. For a standard geoprocessing workflow, the GP tasks are executed in

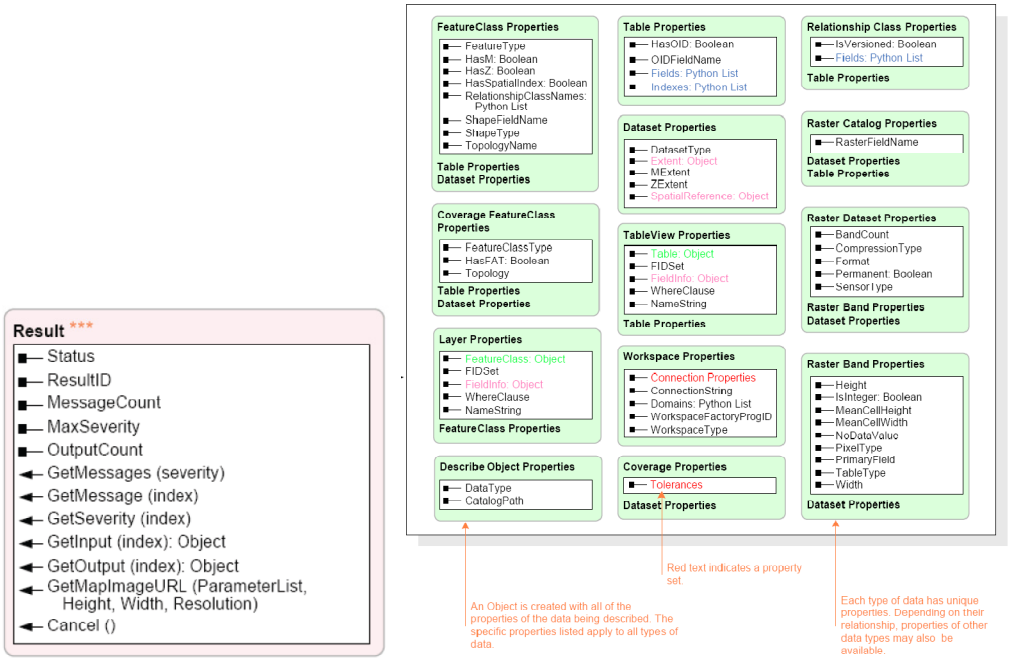

Figure 7 Result Object (left) and Describe Object (right) model diagrams, from ArcGIS 9.3 Geoprocessor Programming Model (Esri 2008). Portions of the Geoprocessor Programming Model reproduced courtesy of Esri. Copyright © 2008 Esri. All rights reserved.

series. Thus, in this case, the recording component will handle and interpret the *Result Object* which is returned from ArcGIS after the execution of each GP task. The *Result Object* is only retained in ArcGIS until the next GP task is executed. Before it is erased, the information contained in the *Result Object* is useful both for saving to the database, and also for directly querying other ArcObjects, in order to obtain additional information relevant to the input/output data elements involved in the GP task. For example, the 'GetInput' method from the *Result Object* returns pointers to the GP task's input data elements. For each input element, the dataset properties *Describe, SpatialReference, and Extent (Objects)* are called upon to obtain information such as data type, spatial projection name and measurement units, and extent coordinates, respectively. Once the provenance information is collected the workflow then executes the next GP task, and again performs the provenance collection operations on the new *Result Object*.

The storage module formats the provenance information into RDF triples using the *RDFlib* library for Python. RDF provides a structure for maintaining the uniqueness of data elements, fosters the descriptive relationships between these elements (be they web pages or other data), and gives semantic meaning to connecting elements. A simple example (3) is provided as follows:

```
Triple 1:   SpatialTask hasInput streets
Triple 2:   streets hasGeometryType Line                    (3)
```

Thus, formatting into RDF triples results in a graph of linked data elements—which, in this implementation, maintain the structure of the geoprocessing workflow via the relationships between GP tasks, input and output data elements, and provenance information (refer to Figure 4). In other words, *workflows* have *tasks*, tasks have *data* (input or output), and data have *properties*—as in the example above (3), the spatial data layer `'streets'` has a *GeometryType* of type `'Line'`, and is input to geoprocessing task `'SpatialTask.'` The RDF triples generated by the storage module are stored in an open-source MySQL relational database using Python. The benefits of this approach are reaped through the integration of the information collection and storage components with Python, along with the persistence and data handling/query capabilities of a relational database.

For querying data in RDF format, the query module constructs SPARQL statements. With the aid of *RDFlib*, the query module interprets and converts the SPARQL statements to Structured Query Language (SQL, Chamberlin and Boyce 1974) queries, which in turn are executed against the MySQL database. The query results are then returned to end users. SPARQL queries are similar to SQL in the use of SELECT and WHERE statements; but, unlike SQL, traverse links between elements to find results that have shared properties. For example, using the RDF example (3), a generalized

query and response to find the spatial data layer that has a `'Line'` *GeometryType* could be (4):

```
Query:      SELECT ?spatialData WHERE ?spatialData hasGeometryType Line
Response:   streets                                                    (4)
```

Although the underlying structure of RDF data is simple, the interlinked nature of the data elements can be complex. SPARQL enables navigation of the semantic relationships between RDF data elements in the spatial provenance information store.

This study's interest lies is in querying the provenance information for semantic links between workflow iterations, tasks, and data properties. Spatial provenance-related questions may be forward-looking ("*Which data or tasks are affected by an update to a source data layer (X)?"),* backward-looking *("What are the input data differences from original results (Y) to updated results (Z)?"*), or may be used to assess the effect of updates in the spatial data processing workflow ("*What is the difference or variability in results for area (A) after different executions of the workflow?*"). Practical questions and queries of interest to geological carbon sequestration research are described in the following chapter.

CHAPTER 5

EXPERIMENTS

This chapter evaluates the benefits of the spatial provenance framework by studying its use in tracking a spatial data transformation workflow in the domain of geological carbon sequestration, and by demonstrating spatial provenance integration with the spatial analytic workflow and decision-making. The first experiment (section 5.1) tests SPARQL queries on the RDF-based spatial provenance store in order to assess the propagation of errors through several data transformation steps in a spatial data handling workflow, while the second experiment (section 5.2) compares similarities and differences in spatial provenance information between alternative instances of the workflow as it evolves and is updated over time.

5.1 SPATIAL ERROR PROPAGATION

The regional CO_2 storage resource for the Mt. Simon Sandstone reservoir within the Illinois Basin[2] was initially estimated in 2008, then later updated in 2009 as new data for the reservoir became available (Figure 8). In this first experiment, we will look at a hypothetical scenario where new work is performed for the next update of the volumetric sequestration estimate, in 2010.

For the purpose of this example, an anomaly has been identified in the results layer `'mts_co2_01c'`—which is the estimated geologic CO_2 storage resource in the Mt. Simon Sandstone reservoir as calculated on 04/11/2010. Starting with this result, we query the provenance information in order to back-track and identify the source data involved in the calculation, and look for possible errors either present in the input data or

[2] The Illinois Basin is an oval-shaped geologic depression covering approximately 60,000 mi^2 in the U.S. midcontinent, principally underlying Illinois, southwestern Indiana, and western Kentucky (Buschbach and Kolata 1990).

introduced during the application of data transformation methods. After finding the input error source, we then find all data 'downstream' which are derived from this errant input data and are spatially affected by the error. The navigation of provenance information chains in the RDF structure enables the discovery of instances where any data 'downstream' of the error need to be corrected and recalculated.

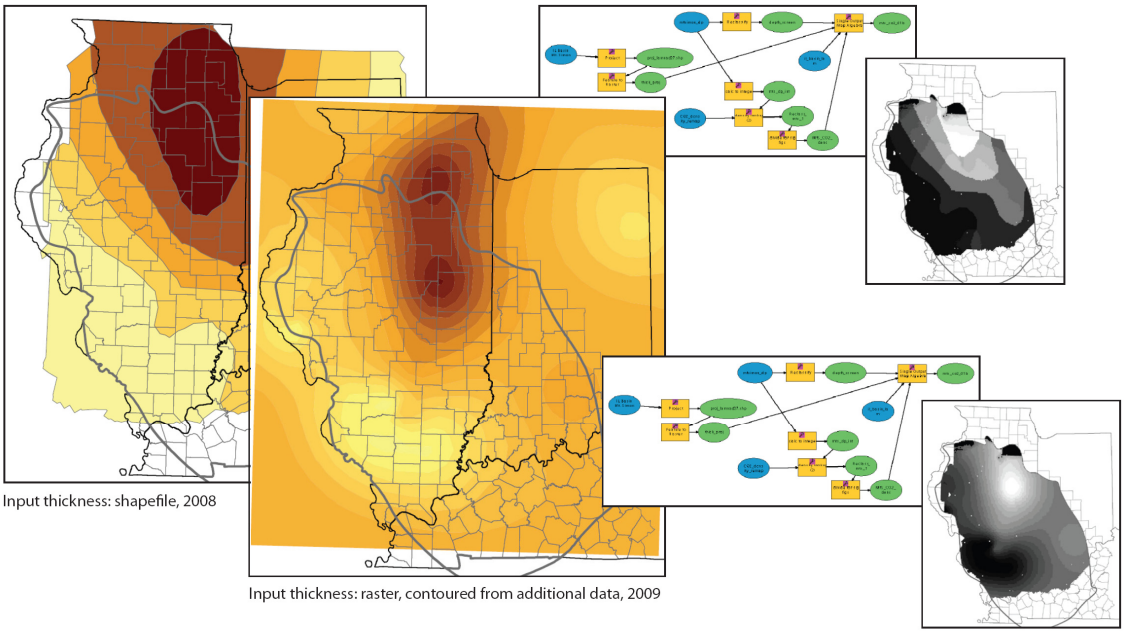

Figure 8 Visualization of an example geological carbon sequestration calculation, showing differences between select input and output data for two different iterations of the workflow, respectively, in 2008 (above), and 2009 (below). *Image sources: "Input thickness: shapefile, 2008" (upper left) modified from (Finley 2005); remaining images, this thesis—created in Esri ArcMap and ModelBuilder.*

To begin, we perform a SPARQL query on the provenance information to back-track and find the data involved in the workflow process. In the query's `'WHERE'` clause (Figure 9), we ask for such parameters as *input data ID*, *name*, and workflow *run date* for all inputs related to the resultant dataset (`'mts_co2_01c'`) dated 04/11/2010. The results of the query (Figure 9) show the four input data elements and the formula parameters used to calculate the volumetric estimation. After reviewing the input calculation and

data, we identify an error in the data layer `'depth_screen'`, which is the input for a *SingleOutputMapAlgebra* geoprocessing task (the Map Algebra raster calculator in ArcGIS). In a similar fashion (not shown in figure), we further trace the error back to this layer's source data, namely to the layer `'mtsimon_dp'`, which is the initial input depth map used to screen for suitable reservoir depths in our geological sequestration example.

```
query = 'PREFIX ns: <http://example.edu/namespace/> \
SELECT ?dataIDin ?input ?taskname ?date \
WHERE { \
        ?script ns:hasTask ?taskname . \
        ?script ns:RunDate ?date . \
        ?task ns:hasInput ?dataIDin . \
        ?task ns:hasOutput ?dataIDout . \
        ?dataIDout ns:Value <ns:data\\mts_co2_01c> . \
        ?dataIDin ns:Value ?input . \
        ?task ns:ToolName ?name \
        FILTER ( \
                ?date = "041110" \
        ) \
}'

5 results

(('ns:GPtask_041110.121350_input_1'), ('ns:7758.171737 * 7758.171737 * mtsimoniso_n * 0.08 *
MtS_CO2_dens * depth_screen * il_basin_lam * 0.01 / 2204.62234'), ('ns:SingleOutputMapAlgebra_sa'))

(('ns:GPtask_041110.121350_input_2'), ('ns:data\\depth_screen'), ('ns:SingleOutputMapAlgebra_sa'))

(('ns:GPtask_041110.121350_input_3'), ('ns:data\\il_basin_lam'), ('ns:SingleOutputMapAlgebra_sa'))

(('ns:GPtask_041110.121350_input_4'), ('ns:data\\MtS_CO2_dens'), ('ns:SingleOutputMapAlgebra_sa'))

(('ns:GPtask_041110.121350_input_5'), ('ns:data\\mtsimoniso_n'), ('ns:SingleOutputMapAlgebra_sa'))
```

Figure 9 SPARQL query (native, in Python) and results: properties of data sources that contributed to a specific, errant, geoprocessing result. (Note that for this and all subsequent SPARQL examples, detailed RDF namespaces, Python *RDFlib* library references for RDF literals and URIs, and pathnames to data have been generalized for presentation).

Next, knowledge of the incorrect data source is used to perform a SPARQL query which finds all 'infected' data which are derived from the incorrect depth layer. We can create queries to find the data immediately derived from a certain geoprocessing task, as well as search for data that is derived further downstream after several successive tasks.

Initially, we look to find the data immediately derived from the geoprocessing task that incorporates `'mtsimon_dp'` as an input data source (Figure 10a). We

subsequently substitute the derived outputs for new inputs in the SPARQL 'WHERE' clause (Figure 10b) to search for data that are derived further downstream after several successive geoprocessing tasks. Thus, in examining the query results from both scenarios, we find that indeed a familiar derived layer ('depth_screen', as identified in Figure 10a) contains the inherited mapping error, but we also find that further down the processing chain is another layer that has been affected (Figure 10b). This additional layer ('mts_co2_dens') is the depth-derived density of CO_2 that could potentially be

a)

```
SELECT ?output
WHERE {
         ?script ns:hasTask ?task .
         ?script ns:RunDate ?date .
         ?task ns:hasInput ?dataIDinput .
         ?dataIDinput ns:Value <ns:data\\mtsimon_dp> .
         ?task ns:hasOutput ?dataIDout .
         ?dataIDout ns:Value ?output
FILTER (
         ?date = "041110"
         )
}


2 results

('ns:data\\depth_screen')

('ns:data\\mts_dp_int')
```

b)

```
SELECT ?output
WHERE {
         ?script ns:hasTask ?task1 .
         ?script ns:RunDate ?date .
         ?task1 ns:hasInput ?dataIDinput .
         ?dataIDinput ns:Value <ns:data\\mtsimon_dp> .
         ?task1 ns:hasOutput ?tmpIDa .
         ?tmpIDa ns:Value ?tmpname .
         ?tmpIDb ns:value ?tmpname .
         ?task2 ns:hasInput ?tmpIDb .
         ?task2 ns:hasOutput ?tmpIDc .
         ?tmpIDc ns:Value ?tmpname2 .
         ?tmpIDd ns:Value ?tmpname2 .
         ?task3 ns:hasInput ?tmpIDd .
         ?task3 ns:hasOutput ?dataIDout .
         ?dataIDout ns:Value ?output
FILTER (
         ?date = "041110"
         )
}


1 result

('ns:data\\mts_co2_dens')
```

Figure 10 Two SPARQL queries and corresponding results: derived data based on an errant input data source, showing a) data immediately derived from the task with source layer 'mtsimon_dp' as input, and b) data derived after three successive tasks 'downstream' (i.e. data output from task 1 is the input data for task 2, etc.)

stored in the subsurface, and is a co-factor in our volumetric equation (2) along with the initial input depth map.

Thus far we have identified the source *map* layer which has included the error. However, if the original errant data source was, for example, a bad point value used in the creation of the map, we could then use the point location to extend the query of Figure 10b to perform spatially-explicit queries, such as "*find all 'infected' data downstream of this input layer, that have this point in their spatial extent*." We can formulate a SPARQL query, then, that looks for derived data that have this error point—located at a hypothetical X,Y coordinate of (3000000, 2000000)—*within* their spatial extents, to look for derived map layers that may (or may not be) spatially related to the data point in question (Figure 11).

```
SELECT ?output ?xmin ?xmax ?ymin ?ymax
WHERE {
        ?script ns:hasTask ?task1 .
        ?script ns:RunDate ?date .
        ?task1 ns:hasInput ?dataIDinput .
        ?dataIDinput ns:Value <ns:data\\mtsimon_dp> .
        ?task1 ns:hasOutput ?tmpIDout1 .
        ?tmpIDout1 ns:Value ?tmpname .
        ?tmpIDin2 ns:Value ?tmpname .
        ?task2 ns:hasInput ?tmpIDin2 .
        ?task2 ns:hasOutput ?tmpIDout2 .
        ?tmpIDout2 ns:Value ?tmpname2 .
        ?tmpIDin3 ns:Value ?tmpname2 .
        ?task3 ns:hasInput ?tmpIDin3 .
        ?task3 ns:hasOutput ?dataIDout .
        ?dataIDout ns:Value ?output ;
                ns:ExtentXmin ?xmin ;
                ns:ExtentXmax ?xmax ;
                ns:ExtentYmin ?ymin ;
                ns:ExtentYmax ?ymax .
        FILTER (
                ?date = "041110" &&
                ?xmin < 3000000 &&
                ?xmax > 3000000 &&
                ?ymin < 2000000 &&
                ?ymax > 2000000
        )
}

1 result

('ns:data\\mts_co2_dens',
('2575000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float'),
('4305000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float'),
('1525000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float'),
('3445000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float'))
```

Figure 11 SPARQL query and results: derived data based on a specific input data source, as in Figure 10b. The derived data contain within their spatial extent the location of an errant source data point. Example shows data derived after three successive tasks, but can be extended for any number of successive 'downstream' geoprocessing tasks.

The result of this search is the same CO_2 density layer (`'mts_co2_dens'`), which implies that the calculation of this layer could have been 'infected' by the erroneous point. This is because in the first instance (Figure 10b) we identified this layer as being simply a derivational 'descendant' of an errant parent input data source by traversing the provenance information for data inputs to and outputs from the chained workflow tasks. However, in the second query instance (Figure 11), we were able to clarify that this same layer is a 'spatial descendant' of the errant parent layer—and, specifically, spatially contains the location of the identified errant data point source within its spatial extent. If, for example, a derived dataset is only a spatial subset of an errant parent layer (e.g. due to layer clipping), then the error may not always spatially affect, or be spatially contained in, the resultant data layer. In such instances, the results may potentially not need to be recalculated—especially for data- and/or processing-intensive workflow operations that may require a significant amount of computing resources or time to complete.

Overall, our spatial provenance framework performs well in this experiment. It provides the ability to track the 'spatial history' of data transformations while provenance information offers insights into the spatial relationships of errors as they propagate through the workflow to end results. The next experiment concentrates on the temporal aspect of data transformation as we compare two instances of a spatial analytical workflow and its associated data as they change over time.

5.2 TEMPORAL COMPARISON

This experiment is designed to examine the same case scenario in which the regional CO_2 storage potential for the Mt. Simon Sandstone reservoir within the Illinois Basin was estimated in 2008, then later updated in 2009 (see Figure 8). Let us suppose that the input data from 2008 has since been lost—and/or that it is too time-consuming to locate and retrieve—but that we still have a record of important spatial provenance

information from this data processing, which had been in place for the 2008 geoprocessing workflow. The recording of spatial provenance incorporated spatial summary statistics which are used to assess the impact of the input data and generate a report for user-defined areas of interest. The provenance information chain is traversed 'upstream' from the results back to the input data. Subsequently, this information is used to compare data which were input into the workflow at different times, in order to gain insights into the nature of the data that were involved in the workflow's previous execution.

To begin comparing the new input data source with the old inputs, we first perform a simple assessment of the differences in spatial provenance information between the alternate *workflows* run at two different times. To do this, we perform a SPARQL query on the provenance information. In the query's `'WHERE'` clause, we ask for such parameters as workflow *run date*, *user comments*, *commands*, and *spatial summary statistics* from all geoprocessing tasks in this workflow that had results for a user-specified county (Figure 12).

The results of the provenance query show that two different sets of geoprocessing tasks were performed—one on 04/09/2008, and a later iteration that was executed on 07/05/2009 (Figure 12). In directly comparing elements of each query result against each other, it is found that there was no change to the basic sequestration volumetric equation (see Equation 2). Yet, the query results show a change in input data layers, and that a new reservoir thickness data layer named `'mtsimon_iso'`, added in 2009, replaced the previous version named `'thick_proj'`. This data update, in turn, supports the corresponding change observed in the spatial summary statistics values for a

```
county = raw_input('results in county: ')

query = 'PREFIX ns: <http://example.edu/namespace/> \
SELECT ?rundate ?comment ?command ?rng ?mean ?sum \
WHERE { \
        ?script ns:RunDate ?rundate ; \
                ns:hasUserComments ?comment ; \
                ns:hasTask ?task . \
        ?task ns:Command ?command ; \
              ns:hasOutput ?dataIDout . \
        ?dataIDout ns:isSpatial ?dataIDspatial . \
        ?dataIDspatial ns:SpatialStat ?stat . \
        ?stat ns:COUNTY_NAM \"%s\" ; \
                ns:RANGE ?rng ; \
                ns:MEAN ?mean ; \
                ns:SUM ?sum \
}' % (county)


2 results

('040908',
'april 2008 trial 3',
'gp.SingleOutputMapAlgebra_sa("10000 * 10000 * thick_proj * 0.08 * MtS_CO2_dens * depth_screen *
il_basin_lam * 0.01 / 2204.62234", mts_co2_01c,"depth_screen;il_basin_lam;MtS_CO2_dens;thick_proj")',
'906971.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float',
'3564080.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float',
'1015760000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float')

('070509',
'atlas 2 summer 2009',
'gp.SingleOutputMapAlgebra_sa("10000 * 10000 * mtsimon_iso * 0.08 * MtS_CO2_dens * depth_screen *
il_basin_lam * 0.01 / 2204.62234", mts_co2_01c,"depth_screen;il_basin_lam;MtS_CO2_dens;mtsimon_iso")',
'656018.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float',
'3504940.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float',
'998908000.0', language=None, datatype='http://www.w3.org/2001/XMLSchema#float')
```

Figure 12 SPARQL query (native, in Python, to show inclusion of 'county' user option) and results: properties for two similar geoprocessing workflows run at different times. The full geoprocessing *command* element is shown in bold type to help clarify the results.

user-specified U.S. county, which show a 35% reduction in the estimated CO_2 storage resource of the Mt. Simon Sandstone from the original estimate based on input data from 2008.

Using a similar SPARQL query (not shown) to determine whether the two data layers identified above are derived data or initial source data, we find that the 2008 input data for the volumetric equation (`'thick_proj'`) was actually derived from a different source layer, named `'MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py.shp'`—a vector data layer. Note that both results from 2008 and 2009, as well as the newer reservoir thickness input from the 2009 calculation, are raster datasets—a difference which implies that a vector-to-raster transformation was part of the workflow in 2008, yet was not incorporated in 2009. We further query the spatial provenance information pertinent to

the initial input data sources (namely, layers 'MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py.shp' and 'mtsimon_iso' for 2008 and 2009, respectively) for such properties as *data type, projection, measurement units*, and *cell size* (Figure 13), thus we can now directly compare temporal differences between the original reservoir thickness *data source* properties which may have an effect on the results.

```
SELECT ?name ?dataID ?type ?ref ?meas ?cellx ?celly \
WHERE { \
         ?dataID ns:Value ?name . \
         ?dataID ns:DataType ?type . \
         ?dataID ns:SpatialReference ?ref . \
OPTIONAL { \
         ?dataID ns:MeasurementUnits ?meas . \
         ?dataID ns:CellWidth ?cellx . \
         ?dataID ns:CellHeight ?celly . \
         } \
FILTER ( \
        regex(str(?name), "MtSimon_Iso_ft_Py") || \
        regex(str(?name), "mtsimon_iso") \
       ) \
}'


2 results

('ns:data\\MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py.shp',
'ns:GPtask_040908.153950_input_1',
'ns:ShapeFile',
'GCS_North_American_1983',
None,
None,
None)

('ns:data\\mtsimon_iso',
'ns:GPtask_070509.160036_input_5',
'ns:RasterDataset',
'NAD_1927_Lambert_Conformal_Conic',
'Foot_US',
'10000.0',
'10000.0')
```

Figure 13 SPARQL query and results: differences between reservoir thickness input data sources for two different versions of the volumetric calculation, in 2008 and 2009, respectively.

In the query results (Figure 13), some differences are observed in the spatial provenance information between the two inputs. For example, the *spatial reference* of the vector-based 2008 data was in Geographic Decimal Degrees, using the 1983 North American Datum (NAD), whereas the data from 2009 was raster-based, and in the Lambert Conformal Conic projection, NAD 1927. The 'OPTIONAL' query parameters for

(ground) *measurement units* do not apply to Geographic data (spherical, measured in degrees), nor are the *cell size* parameters applicable to vector data. These are basic differences gleaned from the provenance information, which reveal how the 2008 result was derived without having access to the original input data from 2008.

The experiment described above serves mainly as a temporal comparison between two alternate geoprocessing workflow instances and their associated input data layers based on spatial provenance. The query examples show how, through provenance information, the workflow evolution can be better understood. The 'summary' spatial statistical information collected is a first-approach toward measuring *differences* in spatial provenance between alternate workflows from different times. Additional spatial provenance information and associated queries help further explore the effects of alternative spatial parameters or transformations upon the derived data. Similarly, additional temporal queries may help researchers see spatial *histories* of the various input or resultant datasets over time, e.g. *cell size* or spatial *extent* histories of a workflow's data components, which, if changed over successive workflow iterations, could offer insights into changing assumptions or scale-related decisions applied to the data.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

This thesis has established a spatial provenance framework for enhancing GIS-based analysis, based on open-source software and semantic web technology and the development of automatic data lineage recording capabilities for desktop GIS software. Through the case study, the thesis has demonstrated the value added to spatial analytic workflow processes within traditional desktop GIS by spatial provenance integration. The experimental examples show that by incorporating semantic relationships and queries built on the RDF data structure, the framework can successfully a) automatically capture, organize, and store provenance information for spatial data and data-derivation tasks within data transformation workflows, b) semantically query and traverse chained datasets and workflow lineage information, and c) compare stored provenance information records both spatially and temporally.

In effect, the spatio-temporal degrees to which errors are propagated from source data through derived results are captured in an implementation of the framework. This is important, particularly to the case study, in the context of results reporting. The geologic CO_2 storage resource estimates may be summed and reported by the varied spatial scales of *county*, *state*, *region*, or potentially by *country* of the world—and for different reporting periods or publications. In the context of errors identified and corrected (or new data acquired), and added to the data processing workflow, exactly *which* results need to be updated, and *where,* affect the time and level of efforts involved with re-processing the results. For example, there may be new data available in an area of the state of Indiana that will have no spatial effect on the next set of updated results for Illinois, or there may have been errors found in a national dataset that may or may not affect different sub-regions, depending on the spatial characteristics of how the data are

used. If the re-calculation for the entire spatial extent is very computationally expensive and we know that only one or a few of the data points are affected, spatial provenance can enable correct identification of the region that is affected and the values that need to be re-calculated. Thus, the framework presented in this thesis can be applied to larger or more complex GIS datasets and workflows, and promises to result in numerous hours of computing resources and human efforts saved.

The Python programming language was used to implement the spatial provenance framework primarily due to its ease of use, its flexibility in communicating with the ArcGIS Geoprocessor and MySQL database, as well as the availability of add-on libraries for supporting RDF data management and SPARQL query interpretation. This implementation in the desktop GIS environment is built on an open provenance recording and management architecture for a specific geological analytical case, and uses customized programming for just one of several GIS software packages available. Yet, this thesis research employs general components (extensible to other calculation workflows in any desktop GIS environment), and principles that can be scaled to massive data sets and network-based collaborative GIS environments that benefit from spatial provenance and semantic query-based access to such provenance information.

Fundamental challenges to further enhancement of the framework lie partly in standardization of the spatial provenance information elements that are captured and stored—which are similar to the issues involved in instituting and maintaining formalized geospatial metadata standards that are implemented and 'enforced', yet flexible and interoperable, across different computing and GIS platforms. Additionally, as the collection of too much provenance information could conceivably slow down automated data processing workflows or data management, too little information would decrease the usefulness of the information store, and thus an 'information content' balance should be strived for.

Another issue is that the recording of spatial provenance information, compounded by numerous users and agencies and institutions performing spatial analytic work, is undeniably diverse and complex. Thus, smaller groupings of provenance information related to specific data layers could be packaged and exported from centralized provenance information stores in RDF/XML (.xml/.txt) format to provide more portability in sharing provenance information for specific derived-data products to consumers of these 'end member' datasets, just as geospatial metadata files sometimes are presented. Additionally, the RDF/XML files can be inspected independently from queries by using RDF 'viewers', which graphically represent the relationships between spatial data and transformations, and facilitate the visual identification of linked information elements (Figure 14). Users collaborating via shared workflows and enterprise data resources would benefit from a central information store where everyone has access to RDF-formatted provenance information and stored SPARQL queries. Although the ability for scientific collaborators and/or colleagues to reproduce, or duplicate, results is a foundation of scientific analysis, it is not ensured simply by the incorporation of provenance information management into data transformation workflows. Consequently, great care must be taken by information producers to properly store and archive any datasets that are the source material for derived information.

In conclusion, a rich set of provenance information benefits systematic assessment of data context and quality when deriving or regularly updating time- and spatially-dependent results. Compared to systems without the capability to capture provenance information (or systems that have limited capabilities), this thesis argues that having access to the spatial provenance of data derived via transformational workflows not only helps users document and keep track of multiple instances of evolving workflows, but also provides them with new ways to: analyze and assess the impact of

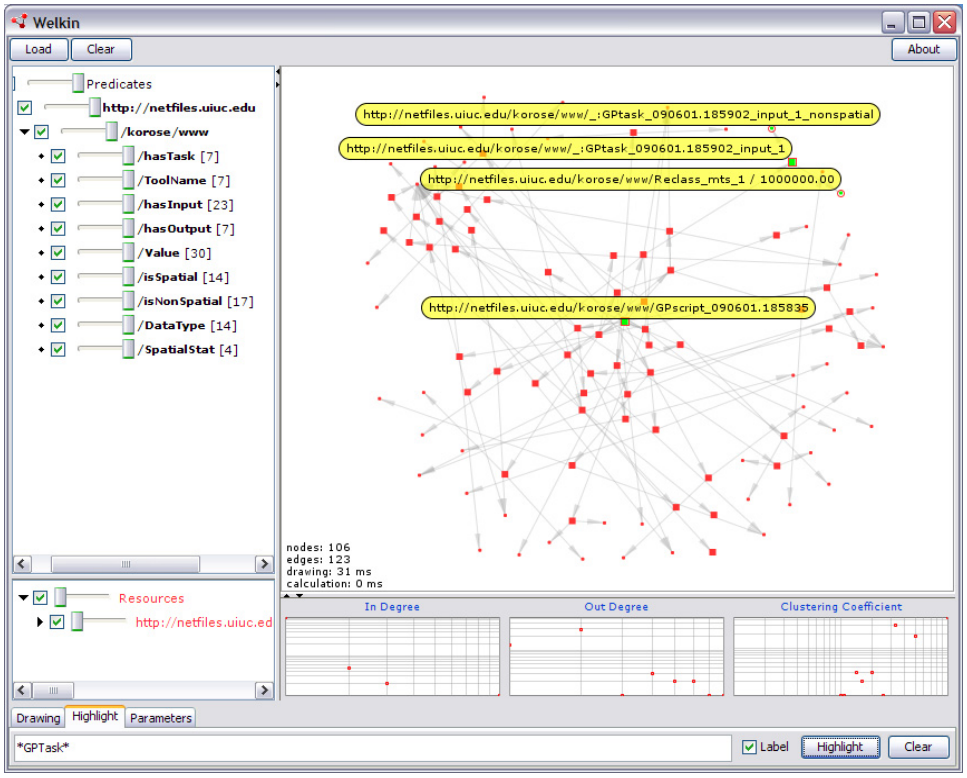

Figure 14 RDF graph visualization created in Welkin v. 1.1 (Massachusetts Institute of Technology 2009) showing interconnections between spatial workflow tasks, input and output data, and various properties of these data. Shown in the graph are the spatial provenance data captured and stored for the experiment presented in section 5.2 in this thesis, which result from the execution of the Python script as detailed in the Appendix.

input data updates or revisions on the derived results, qualify the results for comparison with alternate instances of the workflow, and determine the appropriate usage of a derived dataset based on in-depth examination of its ancestral data sources. This research contends that by introducing spatial provenance management into spatial analytical processes, findings are more robust than in spatial analytical workflows without provenance support.

The research of this thesis has advanced the knowledge of spatial provenance by demonstrating the benefits of adding provenance capabilities to GIS-based analysis and spatial data transformations. Further work continuing this direction of spatial provenance research could involve refining the spatial provenance elements and queries, providing a

robust and flexible user-friendly interface for managing query and results information, or extending this implementation's spatial provenance information management and dissemination capabilities to cyberinfrastructure-based data transformation for collaborative GIS-based work.

REFERENCES

Alonso, G., and Hagen, C., 1997. Geo-Opera: Workflow Concepts for Spatial Processes. Advances in Spatial Databases, 5th International Symposium, SSD '97 Conference Proceedings. Lecture Notes in Computer Science Vol. 1262, 15-18 July 1997. Berlin, Germany: Springer, 238-258.

ARIS, 2005. *ArisFlow 2.4*. Available from: http://www.aris.nl/arisflow/pag/en/index.html [Accessed 26 March 2008].

Bachman, C.W., 1969. Data Structure Diagrams. *Data Base* 1 (2), 4-10.

Bose, R., and Frew, J., 2004. Composing Lineage Metadata with XML for Custom Satellite-Derived Data Products. 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece.

Bose, R., and Frew, J., 2005. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys* 37 (1), 1-28.

Buneman, P., Khanna, S., and Tan, W.C., 2001. Why and Where: A Characterization of Data Provenance. Proceedings of the International Conference on Database Theory (ICDT '01), January 2001. London, UK: Springer, 316-330.

Buschbach, T.C., and Kolata, D.R., 1990. Regional Setting of Illinois Basin. *In:* M.W. Leighton, D.R. Kolata, D.F. Oltz and J. J. Eidel, eds. *Interior Cratonic Basins, AAPG Memoir 51*. Tulsa: American Association of Petroleum Geologists, 29-55.

Chamberlin, D.D., and Boyce, R.F., 1974. SEQUEL: A Structured English Query Language. Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control, 1-3 May 1974. Ann Arbor, Michigan: Association for Computing Machinery, 249-264.

Chen, P.P., 1976. The Entity-Relationship Model-- Toward a Unified View of Data. *ACM Transactions on Database Systems* 1 (1), 9-36.

Chen, P.P., 1977. The Entity-Relationship Model: Basis for Enterprise View of Data. *Proceedings IFIPS* (46), 76-84.

Clarke, D.G., and Clark, D.M., 1995. Lineage. *In:* S. C. Guptill and J. L. Morrison, eds. *Elements of Spatial Data Quality*. Oxford: Elsevier Science, 13-30.

Clifford, B., Foster, I., Voeckler, J.S., Wilde, M., and Zhao, Y., 2008. Tracking Provenance in a Virtual Data Grid. *Concurrency and Computation: Practice and Experience* 20 (5), 565-575.

eBank UK, 2008. *eBank UK Study of Provenance* [online]. UKOLN, University of Bath. Available from: http://www.ukoln.ac.uk/projects/ebank-uk/provenance [Accessed 17 February 2008].

Esri, 1982. *ARC/INFO Geographic Information System (GIS)*. Redlands, CA: Esri.

Esri, 1999. *ArcGIS 8.0*. Redlands, CA: Esri.

Esri, 2000a. ModelBuilder for ArcView Spatial Analyst 2, An ESRI White Paper. Redlands, CA. Available from: http://www.esri.com/library/whitepapers/pdfs/model_bldravsa2.pdf

Esri, 2000b. Developing Applications with ArcInfo: An Overview of ArcObjects, An ESRI White Paper. Redlands, CA. Available from: http://www.edgetech-us.com/Pdf/ai8_arcobjects.pdf

Esri, 2002. Modeling and Using History in ArcGIS, ESRI Technical Paper. Redlands, CA. Available from: http://downloads2.esri.com/support/whitepapers/ao_/Modeling_and_Using_History_in_ArcGIS.pdf

Esri, 2007. Job Tracking for ArcGIS Workflow Management Solution. Redlands, CA. Available from: http://www.esri.com/library/whitepapers/pdfs/jtx-workflow-mgmt.pdf.

Esri, 2008. ArcGIS 9.3 Geoprocessor Programming Model, 9.3-version Geoprocessor. Available from: http://webhelp.esri.com/arcgisdesktop/9.3/pdf/Geoprocessor_93.pdf

Esri, 2009. *ArcGIS Desktop 9.3 Help, About Raster Dataset Properties.* [online]. Esri. Available from: http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=About_raster_dataset_properties [Accessed 18 April 2010].

Federal Geographic Data Committee, 1998. Content Standard for Digital Geospatial Metadata FGDC-STD-001-1998 (revised June): Federal Geographic Data Committee, Washington, D.C.

Finley, R., principal investigator, 2005. An Assessment of Geological Carbon Sequestration Options in the Illinois Basin, Final Report to U.S. Department of Energy October 1, 2003–September 30, 2005. Available from: http://sequestration.org/publish/phase1_final_rpt.pdf

Fonseca, F., Egenhofer, M., Agouris, P., and Câmara, G., 2002. Using Ontologies for Integrated Geographic Information Systems. *Transactions in GIS* 6 (3), 231-257.

Frew, J., and Bose, R., 2001. Earth System Science Workbench: A Data Management Infrastructure for Earth Science Products. In Proceedings of the 13th International Conference on Scientific and Statistical Database Management, George Mason University, 18-20 July, 2001. Fairfax, Virginia: IEEE Computer Society, 180-189.

Gahegan, M., and Pike, W., 2006. A Situated Knowledge Representation of Geographical Information. *Transactions in GIS* 10 (5), 727-749.

Geographic Designs, 1993. *Geolineus Version 3.0 User Manual*. Santa Barbara, CA.

Geosciences Network, 2002. GEOsciences Network (GEON): National Science Foundation, Information Technology Research Grant. Original edition, EAR 0225421. Available from: http://www.geongrid.org

Grossner, K.E., Goodchild, M.F., and Clarke, K.C., 2008. Defining a Digital Earth System. *Transactions in GIS* 12, 145-160.

International Organization for Standardization, 2003. ISO 19115:2003, Geographic information -- Metadata, edited by Technical Committee 211 -- Geographic information/Geomatics.

Keller, R.M., 1998. *Testing Whether a Graph is Acyclic* [online]. Harvey Mudd College. Available from: http://www.cs.hmc.edu/~keller/courses/cs60/s98/examples/acyclic/ [Accessed April 2008].

Klyne, G., Carroll, J., and McBride, B., 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation: World Wide Web Consortium (W3C). Available from: http://www.w3.org/TR/rdf-concepts/

Lake, R., and Cuthbert, A., 2000. Geography Markup Language (GML) v1.0: OGC Recommendation Paper 00-029: Open Geospatial Consortium (OGC). Available from: http://portal.opengeospatial.org/files/?artifact_id=7197

Lanter, D., 1992. GEOLINEUS: Data Management and Flowcharting for ARC/INFO. In *Technical Software Series S-92-2*. Santa Barbara, CA: National Center for Geographic Information and Analysis.

Lanter, D.P., 1989. Techniques and Methods of Spatial Data-Base Lineage Tracing. Dissertation (PhD), University of South Carolina, Columbia.

Lanter, D.P., and Veregin, H., 1990. A Lineage Meta-Database Program for Propagating Error in Geographic Information Systems. Proceedings of the GIS/LIS Conference, 7-10 November 1990. Anaheim, CA: American Society for Photogrammetry and Remote Sensing, American Congress on Surveying and Mapping/Association of American Geographers/Urban and Regional Information Systems Association/AM/FM International, 144-153.

Lanter, D.P., 1991. Design of a Lineage-Based Meta-Database for GIS. *Cartography and Geographic Information Systems* 18 (4), 255-261.

Lanter, D.P., and Veregin, H., 1992. A Research Paradigm for Propagating Error in Layer-Based GIS. *Photogrammetric Engineering and Remote Sensing* 58 (6), 825-833.

Lanter, D.P., 1993. A Lineage Meta-Database Approach Toward Spatial Analytic Database Optimization. *Cartography and Geographic Information Systems* 20 (2), 112-121.

Lanter, D.P., 1994. A Lineage Metadata Approach to Removing Redundancy and Propagating Updates in a GIS Database. *Cartography and Geographic Information Systems* 21 (2), 91-98.

Luo, J., 2007. The Semantic Geospatial Problem Solving Environment: An Enabling Technology for Geographical Problem Solving Under Open, Heterogeneous Environments. Thesis (PhD), The Pennsylvania State University.

Massachusetts Institute of Technology, 2009. *Welkin 1.1*. Available from: http://simile.mit.edu/welkin/ [Accessed 14 April 2009].

Medeiros, C.B., and Jomier, G., 1993. Managing Alternatives and Data Evolution in GIS. ACM-ISCA Workshop on Advances in Geographic Information Systems, November 1993. Baltimore, MD.

Medeiros, C.B., and Jomier, G., 1994. Using Versions in GIS. Database and Expert Systems Applications, 5th International Conference, DEXA '94, Proceedings. Lecture Notes in Computer Science Vol. 856, 7-9 September 1994. Athens, Greece: Springer, 465-474.

Medeiros, C.B., Bellosta, M.J., and Jomier, G., 1996. Managing Multiple Representations of Georeferenced Elements. Seventh International Workshop on Database and Expert Systems Applications, DEXA 1996, 9-10 September 1996. Zurich, Switzerland: EEE-CS Press, 364-370.

Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., and Paulson, P., 2007. The Open Provenance Model (v1.00). Technical Report, University of Southampton. Available from: http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf

National Science Foundation (NSF), *Position Papers from the NSF Workshop on Cyber-GIS* [online]. NSF TeraGrid Science Gateway Program and the Office of Cyberinfrastructure. Available from: http://www.cigi.illinois.edu/cybergis/participants.php [Accessed April 2010].

Nyerges, T.L., 1989. Schema Integration Analysis for the Development of GIS Databases. *International Journal of Geographical Information Systems* 3 (2), 153-183.

Oracle Corporation, 2008. *MySQL Community Server, Relational Database Management System, version 5.1.32*. Available from: http://www.mysql.com/ [Accessed 12 November 2008].

Patel-Schneider, P., Hayes, P., and Horrocks, I., 2004. OWL Web Ontology Language: Semantics and Abstract Syntax, W3C Recommendation: World Wide Web Consortium (W3C). Available from: http://www.w3.org/TR/owl-semantics/

Prud'hommeaux, E., and Seaborne, A., 2008. SPARQL Query Language for RDF, W3C Recommendation: World Wide Web Consortium (W3C). Available from: http://www.w3.org/TR/rdf-sparql-query/

Renaud, K., 2008. *Data Provenance and Annotation Resource Home Page* [online]. Department of Computer Science, University of Glasgow. Available from: http://www.dcs.gla.ac.uk/~karen/Provenance [Accessed 17 February 2008].

Schuurman, N., and Leszczynski, A., 2006. Ontology-Based Metadata. *Transactions in GIS* 10 (5), 709-726.

Simmhan, Y.L., Plale, B., and Gannon, D., 2005. A Survey of Data Provenance Techniques, Technical Report IUB-CS-TR618, Computer Science Department, Indiana University. Available from: http://www.cs.indiana.edu/~ysimmhan/l/pubs/simmhan-tr618-2005.pdf

Spery, L., Claramunt, C., and Libourel, T., 1999. A Lineage MetaData Model for the Temporal Management of a Cadastre Application. 10th International Workshop on Database & Expert Systems Applications, 1-3 September 1999. Florence, Italy.

Spery, L., Claramunt, C., and Libourel, T., 2001. A Spatio-Temporal Model for the Manipulation of Lineage Metadata. *Geoinformatica* 5 (1), 51-70.

Tobler, W.R., 1979. A Transformational View of Cartography. *The American Cartographer* 6 (2), 101-106.

Tomlin, C.D., and Berry, J.K., 1979. A Mathematical Structure for Cartographic Modeling in Environmental Analysis. Proceedings of the 39th Symposium of the American Congress on Surveying and Mapping, 18-24 March 1979. Washington, DC: American Congress on Surveying and Mapping, 269-283.

U.S. Department of Energy Office of Fossil Energy - National Energy Technology Laboratory, 2007. Carbon Sequestration Atlas of the United States and Canada. Available from: http://www.netl.doe.gov/technologies/carbon_seq/refshelf/atlas/

U.S. Department of Energy Office of Fossil Energy - National Energy Technology Laboratory, 2008. Carbon Sequestration Atlas of the United States and Canada, Second Edition. Available from: http://www.netl.doe.gov/technologies/carbon_seq/refshelf/atlasII/

van der Meer, L.G.H., 1992. Investigations Regarding the Storage of Carbon Dioxide in Aquifers in The Netherlands. *Energy Convers. Mgmt* (33), 611.

van Rossum, G., 1995. Python Tutorial, Technical Report CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI).

Veregin, H., and Lanter, D.P., 1995. Data-Quality Enhancement Techniques in Layer-Based Geographic Information-Systems. *Computers Environment and Urban Systems* 19 (1), 23-36.

Vert, G., Stock, M., Jankowski, P., and Gessler, P., 2002. An Architecture for the Management of GIS Data Files. *Transactions in GIS* 6 (3), 259.

Wang, S., Padmanabhan, A., Myers, J., Tang, W., and Liu, Y., 2008. Towards Provenance-Aware Geographic Information Systems. Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 5-7 November 2008. Irvine, CA: ACM GIS, 483.

Wang, S., and Liu, Y., 2009. TeraGrid GIScience Gateway: Bridging Cyberinfrastructure and GIScience. *International Journal of Geographical Information Science* 23 (5), 25.

Wang, S., 2010. A CyberGIS Framework for the Synthesis of Cyberinfrastructure, GIS, and Spatial Analysis. *Annals of the Association of American Geographers* 100 (3), 535-557.

Yue, P., Gong, J., and Di, L., 2009. Augmenting Geospatial Data Provenance Through Metadata Tracking in Geospatial Service Chaining. *Computers & Geosciences* 36 (3), 270-281.

Zhao, Y., Wilde, M., and Foster, I., 2006. Applying the Virtual Data Provenance Model. *In:* L. Moreau and I. Foster, eds. Provenance and Annotation of Data: International Provenance and Annotation Workshop, IPAW 2006, Revised Selected Papers. Lecture Notes in Computer Science, Vol. 4145. Berlin: Springer-Verlag, 148-161.

APPENDIX A: GLOSSARY OF SELECTED TERMS AND ACRONYMS USED IN THIS THESIS

DAG (Directed Acyclic Graph)—graphical representation of an ordered arrangement of linked elements.

Data provenance—provenance information pertaining to data and data properties (as compared to 'process-level provenance').

Derived data—new data created as a result of combining or transforming original (or 'source', 'input') data.

GP, Geoprocessor—component for data-handling and transformational tasks in Esri's ArcGIS software.

MySQL—an open-source, relational database management system.

Process-level provenance—provenance information pertaining to a transformational process or task (as compared to 'data provenance').

Python—an open-source, interpreted computer programming language**.**

RDF (Resource Description Framework)—a structured data/information format consisting of 'triples' of interlinked information, where each triple is of the form <subject, object, predicate>.

RDFlib—library of specialized Python utilities for interacting with RDF-structured data.

SPARQL—a query language designed specifically for RDF data.

SQL (Structured Query Language)—a database querying language.

Task—a GIS data-handling process (or 'command', 'tool', 'operation') performed on one or more data inputs to derive corresponding one or more data outputs.

Workflow—a set of interdependent GIS tasks invoked in an ordered series of steps (sometimes referred to as a 'geoprocessing model'), and often represented as DAG.

APPENDIX B: PYTHON SCRIPT

The following Python script was used in this study's Experiment 5.2. The script runs the spatial analytical workflow for the geological sequestration case study, using the 'new' raster input for geologic reservoir thickness from 2010. The script uses the ArcGIS 9.3-version Geoprocessor along with Python libraries to 1) run the spatial workflow, 2) capture all spatial provenance information in RDF format, 3) store this information in a MySQL database, 4) provide status reports and informative messages to a user in the PythonWin Interactive Window during program runtime, and 5) write spatial provenance information to a RDF/XML text file, if desired by the user, for further analysis outside of the Python scripting environment.

```
# ---------------------------------------------------------------------------
# __SCRIPT8_3_rasternew_compound.py
# Created on: Thu Jan 29 2009 07:45:53 PM
#    modified in PythonWin - Jan 29 2009 to date: CPK
#
#
# This Python script uses the ArcGIS 9.3-version Geoprocessor's
# Result and Describe Objects to capture provenance information
# about the spatial analytical workflow and ArcGIS data processing
# commands.
#
# Spatial provenance information is stored as RDF data in a MySQL
# database.
#
# This script assumes MySQL is installed and configured for RDF data,
# and that RDFlib libraries for Python are available on the local machine.
#
#
# ---------------------------------------------------------------------------


################################################
############ SCRIPT INITIALIZATION
### import system modules
import sys, string, os, time, arcgisscripting, traceback
from dbfpy import dbf

### rdflib imports
import rdflib
from rdflib.Graph import Graph
from rdflib import plugin
from rdflib.store import Store, NO_STORE, VALID_STORE
from rdflib import Namespace
from rdflib import Literal
from rdflib import URIRef
```

```
print "starting script... \n"
print


### prompt user for write output
write_rdf = raw_input('Write file to rdfxml?  (y | n): ')

### prompt for specific comments on this run
user_comment = raw_input('enter script run comments: ')
print user_comment


### variable initialization for test of custom statistics
final_outputID = ""


#######################################
### ARCGIS INITIALIZATION
### Create the Geoprocessor object
gp = arcgisscripting.create(9.3)
gp.overwriteoutput = 1

### set workspace
gp.workspace = "C:/cpk_lap/_grad/arcdir/data"
gp.AddMessage("TEST OF GP.ADDMESSAGE") #use this for when running as GP tool
print "workspace: " + gp.workspace
print

### Check out any necessary ArcGIS licenses
gp.CheckOutExtension("spatial")

### Load required ArcGIS toolboxes...
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx") #
"_analysis"
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Spatial Analyst
Tools.tbx") #_sa"
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Conversion Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management
Tools.tbx")


#def printSpatialProvenance()  # function for later to condense duplicate code


#######################################
### CREATE RDF STORE IN MYSQL

default_graph_uri = "http://rdflib.net/rdfstore"

### set connection string
connect = "host=localhost,user=rdf_user,password=rdf_user,db=rdf_test"
store = plugin.get('MySQL', Store)('rdf_test')


### Open previously created store, or create it if it doesn't exist yet
rt = store.open(connect,create=False)
if rt == NO_STORE:
    #### There is no underlying MySQL infrastructure, create it
    store.open(connect,create=True)
else:
    assert rt == VALID_STORE,"The underlying store is corrupted"

#### There is a store, use it
graph = Graph(store, identifier = URIRef(default_graph_uri))

#### set generic namespace for testing
rdflib = Namespace('http://netfiles.uiuc.edu/korose/www/')
XSD_NS = Namespace('http://www.w3.org/2001/XMLSchema#')
```

```
### show existing data
print "Initial triples in graph: ", len(graph)
print



######################################################################
############################## BEGIN … PROVENANCE CAPTURE AND STORAGE
########
### set time stamp variables
scriptRuntime = time.strftime("%m%d%y.%H%M%S")
scriptID = "GPscript_" + scriptRuntime

print 'runtime = ' + scriptRuntime
tmpdt = scriptRuntime.split(".")
scriptDate = tmpdt[0]
print 'date = ' + scriptDate
scriptTime = tmpdt[1]
print 'time = ' + scriptTime

### Add data to RDF graph as triples
graph.add((rdflib[scriptID], rdflib['RunDate'], Literal(scriptDate)))
graph.add((rdflib[scriptID], rdflib['RunTime'], Literal(scriptTime)))
graph.add((rdflib[scriptID], rdflib['hasUserComments'], Literal(user_comment)))
#graph.add((rdflib['GPscript_' + scriptRuntime], rdflib['hasGeoprocVersion'],
Literal(geoprocversion)))


#######
### ARCGIS LOOP THROUGH RESULT OBJECT INFORMATION
#######
### define main function to work on result object

def process_result(r):
    global final_outputID

    #taskRuntime = time.strftime("%m%d%y.%H%M%S", time.localtime())
    taskRuntime = time.strftime("%m%d%y.%H%M%S")
    taskID = "GPtask_" + taskRuntime

    ### write rdf triple to mysql -- using URI instead of literal
    graph.add((rdflib[scriptID], rdflib['hasTask'], rdflib[taskID]))
    ### print messages for pywin debug and follow-along
    print
    print "***********************************************"
    print "*** GEOPROCESSING TASK (time_id) = " + taskRuntime
    print "***********************************************"
    print


    ### get geoprocessing command string for GP usage from python command
traceback
    print "*** STUFF FROM PYTHON TRACEBACK"
    trace = traceback.extract_stack()
    print trace[3]  #?? is it always 3 ??
    ## WARNING ##################################################
    ## CAN'T OBTAIN THE GP.COMMAND IN THE TRACE IF YOU USE: result = GP.addmessage
    ## HAVE TO USE: process_result(GP.command)
    ## see gp.task geoprocessing code section, after code for process_result()

    print "GEOPROCESSING COMMAND:"
    start = str(trace[3]).rfind("gp.")
    endbit = str(trace[3])[start:]
    command = endbit[:-3]
    ### write to RDF
    graph.add((rdflib[taskID], rdflib['Command'], Literal(command)))
    ## above uses literal for command instance...
    ## if use URI instead, then will not save duplicate commands (tool re-runs):
    ## graph.add((rdflib[taskID], rdflib['Command'], rdflib[command]))
    print command
    print
```

```
    ### GET TOOL NAME
    print "TOOL:"
    end = command.find("(")
    tool = command[3:end]
    ### write to RDF
    graph.add((rdflib[taskID], rdflib['ToolName'], rdflib[tool]))
    print tool
    print

    print "GP.COMMAND USAGE:"
    usage = eval("gp.usage('"+tool+"')")
    print usage
    print

    print
    print "*** GENERAL STUFF FROM RESULT OBJECT"
    print
    print "geoprocessing messages:"
    for i in range (0,r.messagecount):
        ### gp.AddMessage() #use this for when running as GP tool pring msgs in
arcmap console
        print "msg " + str(i) + ": " + r.getmessage(i)
    print


    ###################################
    ### OBTAIN INPUT/OUTPUT INFORMATION, LOOP THROUGH RESULT OBJECT DATA,
    ### TASKS CAN HAVE MULTIPLE INPUTS OR OUTPUTS…

    ### GET INPUTS
    ## would have to convert count to string if URI: rdflib[str(r.outputcount)]
    graph.add((rdflib[taskID], rdflib['InputCount'], Literal(r.inputcount)))
    print "inputs:"
    #inputdata test graph.add((rdflib[taskID], rdflib['hasData'],
rdflib['inputData']))

    for i in range (0,r.inputcount):
        if not r.getinput(i) == "":

            ### added 4.10.10 for compound input concatenation
            new_r = r.getinput(i)
            r_comp = new_r.split(";")
            jmax = 0
            if len(r_comp) > 1:
                print 'compound input'

            for j in range (0,len(r_comp)):
                jmax = jmax + j
                dataID = taskID + '_input_' + str(i+jmax+1) #i + 1 for pretty
number

                ### gptask hasinput <blank>
                graph.add((rdflib[taskID], rdflib['hasInput'],
rdflib['_:'+dataID]))

                ## make inputs URIs so that we can link them ??
                ## old literal test graph.add((rdflib2['inputData'],
rdflib2['hasValue'], Literal(r.getinput(i))))
                ## blank hasvalue datavalue
                graph.add((rdflib['_:'+dataID], rdflib['Value'],
rdflib[r_comp[j]]))

                print r_comp[j]
    print


    ### GET OUTPUTS
    ##
    graph.add((rdflib[taskID], rdflib['OutputCount'], Literal(r.outputcount)))
```

```
    print "outputs:"
    for i in range (0, r.outputcount):
        dataID = taskID + '_output_' + str(i+1)
        graph.add((rdflib[taskID], rdflib['hasOutput'], rdflib['_:'+dataID]))
        graph.add((rdflib['_:'+dataID], rdflib['Value'], rdflib[r.getoutput(i)]))
        print r.getoutput(i)
    print


    ### GET DETAILED INFORMATION FROM INPUTS, LOOP THROUGH DATA AND USE DESCRIBE
OBJECT
    print "----DETAIL-------------"
    print
    print "*** INPUTS:"
    print
    for i in range (0,r.inputcount):

        if not r.getinput(i) == "":

            #added 4.10.10 for compound input concatenation
            new_r = r.getinput(i)
            r_comp = new_r.split(";")
            jmax = 0
            if len(r_comp) > 1:
                print 'compound input'

            for j in range (0,len(r_comp)):
                ##same as above... can condense later
                jmax = jmax + j
                dataID = taskID + '_input_' + str(i+jmax+1) #i + 1 for pretty
number
                print dataID
                print r_comp[j]

                ### Create a describe object
                ### only works for spatial data
                ### use try/except to catch this
                try:
                    ### assumes the in/out data still reside on the system
                    ### therefore might not work in server???
                    #desc = gp.Describe(r.getinput(i))
                    desc = gp.Describe(r_comp[j])
                    ### fails if not spatial data

                    #printSpatialProvenance()  #LATER MOVE CODE TO HERE
                    ## dataID isSpatial dataID_spatial
                    dataID_spatial = dataID + '_spatial'
                    graph.add((rdflib['_:'+dataID], rdflib['isSpatial'],
rdflib['_:'+dataID_spatial]))


                    ############################################################
                    ### GET DATASET PROPERTIES (print for review AND WRITE TO RDF)
                    print "***"
                    print "INPUT " + str(i+jmax+1) + ": from describe object
(getinput):"

                    ## drop spatial ID for now?
                    #graph.add((rdflib['_:'+dataID_spatial], rdflib['DataType'],
rdflib[desc.DataType]))
                    graph.add((rdflib['_:'+dataID], rdflib['DataType'],
rdflib[desc.DataType]))
                    print desc.DataType

                    print desc.CatalogPath


                    ### RASTER DATA PROPERTIES
                    if desc.DataType == "RasterDataset":
                        ## do i need to create a raster?  NO!
                        ## raster = str(desc.CatalogPath)
```

```
                        ## newdesc = gp.describe(raster)
                        print "Format: " + desc.Format
                        print "Cellsize X: " + str(desc.MeanCellWidth)
                        print "Cellsize Y: " + str(desc.MeanCellHeight)
                        graph.add((rdflib['_:'+dataID],
rdflib['RasterFileFormat'], Literal(desc.Format)))
                        graph.add((rdflib['_:'+dataID], rdflib['CellWidth'],
Literal(desc.MeanCellWidth,XSD_NS+u'float')))
                        graph.add((rdflib['_:'+dataID], rdflib['CellHeight'],
Literal(desc.MeanCellHeight,XSD_NS+u'float')))
                    print


                    ### SPATIAL REFERENCE PROPERTIES
                    print "SPATIALREF from spatial reference object (describe
input):"
                    spatialref = desc.SpatialReference
                    print spatialref.Name
                    graph.add((rdflib['_:'+dataID], rdflib['SpatialReference'],
Literal(spatialref.Name)))

                    if not (spatialref.LinearUnitName == ""):
                        print spatialref.LinearUnitName
                        graph.add((rdflib['_:'+dataID],
rdflib['MeasurementUnits'], Literal(spatialref.LinearUnitName)))

                    else:
                        print spatialref.SpheroidName
                        print spatialref.DatumName
                        print spatialref.AngularUnitName
                    print


                    ### EXTENT INFORMATION
                    extent = desc.Extent
                    print "EXTENT from extent object (describe input):"
                    print extent.xmin, extent.ymin, extent.xmax, extent.ymax
                    print
                    #test of extent RDF for paper example 2c
                    #don't use dataID_spatial?  is ID overkill...
                    graph.add((rdflib['_:'+dataID], rdflib['ExtentXmin'],
Literal(extent.xmin,XSD_NS+u'float')))
                    graph.add((rdflib['_:'+dataID], rdflib['ExtentYmin'],
Literal(extent.ymin,XSD_NS+u'float')))
                    graph.add((rdflib['_:'+dataID], rdflib['ExtentXmax'],
Literal(extent.xmax,XSD_NS+u'float')))
                    graph.add((rdflib['_:'+dataID], rdflib['ExtentYmax'],
Literal(extent.ymax,XSD_NS+u'float')))

                except RuntimeError:
                    #dataID isNonSpatial dataID_nonspatial
                    graph.add((rdflib['_:'+dataID], rdflib['isNonSpatial'],
rdflib['_:'+dataID+"_nonspatial"]))
                    print "input " + str(i+jmax+1) + " is not spatial dataset."
                    print
    print


    ### GET DETAILED INFORMATION FROM OUTPUTS, LOOP THROUGH DATA AND USE DESCRIBE
OBJEC
    ##  this is the redundant code that can be condensed later to use same
    ##  code for both inputs and outputs…  was easier to copy here for testing and
keep separate.
    print "*** OUTPUTS:"
    print
    for i in range (0, r.outputcount):
        dataID = taskID + '_output_' + str(i+1) #i + 1 for pretty number

        try:
            desc = gp.Describe(r.getoutput(i))   ### MAKE SURE THIS IS OUTPUT!
```

```
            dataID_spatial = dataID + '_spatial'
            graph.add((rdflib['_:'+dataID], rdflib['isSpatial'],
rdflib['_:'+dataID_spatial]))


            ###############################################################
            ### GET DATASET PROPERTIES (print for review AND WRITE TO RDF)
            print "***"
            print "OUTPUT " + str(i) + ": from describe object (getoutput):"

            #drop spatial for now?
            #graph.add((rdflib['_:'+dataID_spatial], rdflib['DataType'],
rdflib[desc.DataType]))
            graph.add((rdflib['_:'+dataID], rdflib['DataType'],
rdflib[desc.DataType]))
            print desc.DataType


            ### RASTER DATA PROPERTIES
            print desc.CatalogPath
            if desc.DataType == "RasterDataset":
                ##do i need to create a raster?  NO!
                ##raster = str(desc.CatalogPath)
                ##newdesc = gp.describe(raster)
                print "Format: " + desc.Format
                print "Cellsize X: " + str(desc.MeanCellWidth)
                print "Cellsize Y: " + str(desc.MeanCellHeight)
                graph.add((rdflib['_:'+dataID], rdflib['RasterFileFormat'],
Literal(desc.Format)))
                graph.add((rdflib['_:'+dataID], rdflib['CellWidth'],
Literal(desc.MeanCellWidth,XSD_NS+u'float')))
                graph.add((rdflib['_:'+dataID], rdflib['CellHeight'],
Literal(desc.MeanCellHeight,XSD_NS+u'float')))
            print

            ### SPATIAL REFERENCE PROPERTIES
            print "SPATIALREF from spatial reference object (describe output):"
            spatialref = desc.SpatialReference
            print spatialref.Name
            graph.add((rdflib['_:'+dataID], rdflib['SpatialReference'],
Literal(spatialref.Name)))

            if not (spatialref.LinearUnitName == ""):
                print spatialref.LinearUnitName
                graph.add((rdflib['_:'+dataID], rdflib['MeasurementUnits'],
Literal(spatialref.LinearUnitName)))

            else:
                print spatialref.SpheroidName
                print spatialref.DatumName
                print spatialref.AngularUnitName
            print


            ### EXTENT INFORMATION
            extent = desc.Extent
            print "EXTENT from extent object (describe output):"
            print extent.xmin, extent.ymin, extent.xmax, extent.ymax
            print
            graph.add((rdflib['_:'+dataID], rdflib['ExtentXmin'],
Literal(extent.xmin,XSD_NS+u'float')))
            graph.add((rdflib['_:'+dataID], rdflib['ExtentYmin'],
Literal(extent.ymin,XSD_NS+u'float')))
            graph.add((rdflib['_:'+dataID], rdflib['ExtentXmax'],
Literal(extent.xmax,XSD_NS+u'float')))
            graph.add((rdflib['_:'+dataID], rdflib['ExtentYmax'],
Literal(extent.ymax,XSD_NS+u'float')))

            ### TEST: SET FINAL OUTPUT FOR CUSTOM STATISTICS
            final_outputID = '_:'+dataID_spatial
            ###
```

```
        ### message for if not spatial dataset
        except RuntimeError:
            graph.add((rdflib['_:'+dataID], rdflib['isNonSpatial'],
rdflib['_:'+dataID+"_nonspatial"]))
            print "output " + str(i) + " is not spatial dataset."
            print
    print


    ### print message to verify RDF data loaded to MYSQL
    print "Triples in graph after add: ", len(graph)
print

### END process_result()



################################################
### BEGIN GEOPROCESSING ACTIONS
###


### SET Local variables...
il_basin_lam = "il_basin_lam"
depth_screen = "depth_screen"
mts_co2_01c = "mts_co2_01c"
thick_proj = "thick_proj"
mtsimon_iso = "mtsimon_iso"
proj_lamnad27_shp = "proj_lamnad27.shp"
mts_dp_int = "mts_dp_int"
MtS_CO2_dens = "mts_co2_dens"
Reclass_mts_1 = "Reclass_mts_1"
MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py =
"MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py.shp"
mtsimon_dp = "mtsimon_dp"
CO2_density_remap = "C:\\arcdir\\cpk_geodatabase.mdb\\CO2_density_remap"
test_table = "test_table.dbf"


### Process: Reclassify...
if gp.exists(depth_screen):
    gp.delete(depth_screen)
process_result(gp.Reclassify_sa(mtsimon_dp, "Value", "0 2500 0;2500 999999 1",
depth_screen, "DATA"))


### Process: calc to integer...
if gp.exists(mts_dp_int):
    gp.delete(mts_dp_int)
process_result(gp.SingleOutputMapAlgebra_sa("int(mtsimon_dp)", mts_dp_int,
"mtsimon_dp"))

### Process: density lookup (2)...
if gp.exists(Reclass_mts_1):
    gp.delete(Reclass_mts_1)
process_result(gp.ReclassByTable_sa(mts_dp_int, CO2_density_remap, "Depth_From",
"Depth_To", "GRID_VAL", Reclass_mts_1, "DATA"))

### Process: divide for sig figs...
if gp.exists(MtS_CO2_dens):
    gp.delete(MtS_CO2_dens)
process_result(gp.SingleOutputMapAlgebra_sa("Reclass_mts_1 / 1000000.00",
MtS_CO2_dens, "Reclass_mts_1"))

#process_result(gp.Project_management(MGSC_DB_MGSC_Saline_MtSimon_Iso_ft_Py,
proj_lamnad27_shp,
"PROJCS['NAD_1927_Lambert_Conformal_Conic',GEOGCS['GCS_North_American_1927',DATUM[
'D_North_American_1927',SPHEROID['Clarke_1866',6378206.4,294.9786982]],PRIMEM['Gre
enwich',0.0],UNIT['Degree',0.0174532925199433]],PROJECTION['Lambert_Conformal_Coni
c'],PARAMETER['False_Easting',2999994.0],PARAMETER['False_Northing',0.0],PARAMETER
```

```
['Central_Meridian',-
89.5],PARAMETER['Standard_Parallel_1',33.0],PARAMETER['Standard_Parallel_2',45.0],
PARAMETER['Latitude_Of_Origin',33.0],UNIT['Foot_US',0.3048006096012192]]",
"NAD_1927_To_NAD_1983_6",
"GEOGCS['GCS_North_American_1983',DATUM['D_North_American_1983',SPHEROID['GRS_1980
',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.01745329251994
33]]"))

### Process: Single Output Map Algebra...
if gp.exists(mts_co2_01c):
    gp.delete(mts_co2_01c)
process_result(gp.SingleOutputMapAlgebra_sa("7758.171737 * 7758.171737 *
mtsimon_iso * 0.08 * MtS_CO2_dens * depth_screen * il_basin_lam * 0.01 /
2204.62234", mts_co2_01c, "depth_screen;il_basin_lam;MtS_CO2_dens;mtsimon_iso"))



########################################################## TESTING ###
#################### TEST OF SPATIAL STATISTICS
###
print "test of spatial stats"
print

if gp.exists(test_table):
    gp.delete(test_table)
#process_result(gp.ZonalStatisticsAsTable_sa('IL_counties.shp', 'COUNTY_NAM',
mts_co2_01c, test_table))
gp.ZonalStatisticsAsTable_sa('IL_counties.shp', 'COUNTY_NAM', mts_co2_01c,
test_table)


dbf_file = dbf.Dbf(gp.workspace+"/"+test_table)


### QUICK TEST ONLY LOADING THREE RECORDS FOR NOW…
### CAN LOAD ALL DATA LATER BY CHANGING THE RANGE IN LOOP
###
i = 0
for i in range (0,3):
    rec = dbf_file[i]

    graph.add((rdflib[final_outputID], rdflib['SpatialStat'],
rdflib['record_'+str(i+1)]))
    print "record number " + str(i+1)

    #rec details
    print dbf_file.fieldNames[0]
    print rec[0]
    graph.add((rdflib['record_'+str(i+1)],
rdflib[dbf_file.fieldNames[0]],Literal(rec[0])))

    print dbf_file.fieldNames[6]
    print rec[6]
    graph.add((rdflib['record_'+str(i+1)],
rdflib[dbf_file.fieldNames[6]],Literal(rec[6])))

    print dbf_file.fieldNames[7]
    print rec[7]
    graph.add((rdflib['record_'+str(i+1)],
rdflib[dbf_file.fieldNames[7]],Literal(rec[7])))

    print dbf_file.fieldNames[9]
    print rec[9]
    graph.add((rdflib['record_'+str(i+1)],
rdflib[dbf_file.fieldNames[9]],Literal(rec[9])))

    print
    i = i + 1


### QUICK TEST add in one result for now: Co = Champaign
```

```
### can loop to add all data later
###
for i in range (0,len(dbf_file)):
    rec = dbf_file[i]
    if rec["COUNTY_NAM"] == "CHAMPAIGN":
        graph.add((rdflib[final_outputID], rdflib['SpatialStat'],
rdflib[final_outputID+'_stats_rec_'+str(4)]))
        print "record number " + str(i+1)

        #rec details
        print dbf_file.fieldNames[0]
        print rec[0]
        graph.add((rdflib[final_outputID+'_stats_rec_'+str(4)],
rdflib[dbf_file.fieldNames[0]],Literal(rec[0])))

        print dbf_file.fieldNames[6]
        print rec[6]
        graph.add((rdflib[final_outputID+'_stats_rec_'+str(4)],
rdflib[dbf_file.fieldNames[6]],Literal(rec[6])))

        print dbf_file.fieldNames[7]
        print rec[7]
        graph.add((rdflib[final_outputID+'_stats_rec_'+str(4)],
rdflib[dbf_file.fieldNames[7]],Literal(rec[7])))

        print dbf_file.fieldNames[9]
        print rec[9]
        graph.add((rdflib[final_outputID+'_stats_rec_'+str(4)],
rdflib[dbf_file.fieldNames[9]],Literal(rec[9])))

        print
        i = i + 1


dbf_file.close()

## end test of spatial stats
#################################################################



# END GEOPROCESSING ACTIONS
#####################################################################




### COMMIT DATA ADDED TO GRAPH IN DATABASE
graph.commit()

### VERIFY DATA WRITTEN
print "Triples in graph after add: ", len(graph)
print

#print "serialized rdf/xml:"
#print
#print graph.serialize()
#print
### or N3
#print graph.serialize(format='n3')


# write serialized text to a file if user wants it (e.g. using Welkin RDF viewer)
if (write_rdf == 'y'):
    outfile = open("c:/documents and settings/chris/my
documents/cpktest_rdfxml.rdf", "w")
    outfile.write(graph.serialize())
    outfile.close()
    print "RDF/XML file for welkin is in ... /my documents/cpktest_rdfxml.rdf"
```

```
###################################
##I don't always want to close the store after script is run…
print "closing store"
graph.close()
store.close
###################################


print "RDF/XML file for welkin is in ... /my documents/cpktest_rdfxml.rdf"
print
print "script end"
print
"#############################################################################"
print
print
```